Universidad Autónoma de Madrid

Facultad de Ciencias
Máster en Matemáticas y Aplicaciones

TRABAJO DE FIN DE MASTER

Dependent Typing through Closed Cartesian Categories

Presentado por:
Pedro Bonilla Nadal

Dirigido por:
Ángel González Prieto

Curso académico 2020-2021

Dependent Typing through Closed Cartesian Categories.

Pedro Bonilla Nadal

Pedro Bonilla Nadal *Dependent Typing through Closed Cartesian Categories.*

Master's thesis.
Academic year 2020-2021.

Thesis supervisor

Dr. Ángel González Prieto
Departamento de Matemáticas
Universidad Autónoma de Madrid

Máster en Matemáticas y Aplicaciones

Universidad Autónoma de Madrid

Contents

Part I

Category Theory

Chapter 1

First Notions of Category Theory

> "The human mind has never invented a labor-saving machine equal to algebra."
>
> *Stephan Banach (1925)*

For us Category Theory is an area of interest on its own rights, rather than merely an elegant tool. Thus, we will introduce this theory with a point of view that emphasize the intuitive ideas behind each notion.

With this objective in mind, we will get into the habit of introducing the first examples even before introducing the formal definitions. Thus, when the formal definition is introduced it becomes meaningful.

The fundamental idea of Category Theory is that many properties can be unified if expressed in arrow diagrams. Intuitively, a diagram is a directed graph, such that each way of going from a node to another are equals. For example, the diagram:

$$\begin{tikzcd}
& b \arrow[dr, "g"] & \\
a \arrow[ur, "f"] \arrow[rr, dashed, "h"] & & c
\end{tikzcd}$$

Means that $f \circ g = h$. This approach to mathematics emphasises at the relationships between elements, rather than at the structure of the elements themselves. In general, dashed lines means that the existence of that particular arrow is uniquely determined by the solid arrow presents in the diagram.

In this first chapter we introduce the notion of category and some useful properties. The principal references for this chapter are [24] and [29]. To show how categories can be used for programming we will illustrate several concepts with the programming language Haskell. Haskell references can be checked in [26].

1.1 Metacategories

We will start by defining a concept independent of the set theory axioms: the concept of *metacategory*. Then, categories will arise from studying these concepts within set theory. We follow [24] for these definitions.

Traditionally, mathematics is based on the set theory. When we start set theory it is not necessary (it is not possible) to define what a set is. It is similar with the concepts of element and belonging, which are basic to set theory. Category theory can also be used to found mathematics. This theory provide meaning based on other concepts such as object, arrow or composition rather than sets or belonging.

Definition 1.1.1. A *metagraph* consist of *objects*: a, b, c.. and *arrows* f, g, h.... There are also two pairings: dom and codom. This pairings assigns each arrow with an object. An arrow f with $\mathrm{dom}(f) = a$ and $\mathrm{codom}(f) = b$ is usually denoted as $f : a \to b$.

Definition 1.1.2. A metacategory is a metagraph with two additional operations and two properties:

- Operations:
  - *Identity*: assigns to each object a an arrow $1_a : a \to a$.
  - *Composition*: assigns to each pair of arrows f, g with $\mathrm{codom}(f) = \mathrm{dom}(g)$ and arrow $g \circ f$ such that the diagram:

$$\begin{tikzcd} & b \arrow[dr, "g"] & \\ a \arrow[ur, "f"] \arrow[rr, dashed, "g\circ f"'] & & c \end{tikzcd}$$

    commutes. The arrow $g \circ f$ is called the *composite* of f and g.

- Properties:
  - *Associative*: given arrows f, g, h, we have that,

$$(g \circ f) \circ h = g \circ (f \circ h).$$

  - *Unit*: given an object a, and arrows f, g such that $\mathrm{dom}(f) = a$ and $\mathrm{codom}(g) = a$, we have that,

$$1_a \circ g = g, \qquad f \circ 1_a = f.$$

In the context of categories and metacategories, arrows are often called *morphisms*.

We have just define what a metacategory is without any need of set and elements. In most cases we will rely in a set theory interpretation of this definitions, as most examples (specially with our interest in computational example) will rely on this theory. Nonetheless, whenever possible, we will define the concepts working only in terms of objects and arrows.

1.2 Set theory categories

Despite being an alternative foundation of mathematics, in general we will work with categories interpreted with notions of set theory. Unless otherwise stated, we will use the Zermelo-Fraenkel-Gödel axiomatics with Axiom of Choice. In this we are going to difference between *small sets* and *classes*. Sets that are not small are called *large sets*. We recommend [21] for more information.

We have no interest in this work in the reformulation of axioms. Note that we have defined a meta-category without the need for notions of sets. In the same way, many definitions and propositions relating to this theory can be considered without the need for notions of sets.

From now on, we will work based on set theory. Despite that, it is useful to take into account that most of the concept here presented can be defined without making use of this Theory.

Definition 1.2.1. A category (resp. graph) is an interpretation of a metacategory (resp. metagraph) within a set theory.

[1]That is, a graph/category is a pair (O, A) where a collection O consisting of all objects as well as a collection A consisting of all arrows. Their elements hold the same properties that objects and arrows hold on metacategories / metagraph.

We will focus on the category. First, we define the function homeset of a category $C = (O, A)$, wrote as $\hom_C$, as the function:

$$\begin{aligned}\hom_C : O \times O &\mapsto \mathcal{P}(A)\\ (a, b) &\mapsto \{f \in A | f : a \to b\}\end{aligned}$$

We will refer to the collection of objects of a category C as $Ob(C)$ and the collection of arrows as $Ar(C)$. When there is no possibility of confusion we will state $c \in C$ meaning either $c \in Ob(C)$ or $c \in Ar(C)$.

Definition 1.2.2. We say that a category is *small* if the collection of objects is given by a set (instead of a proper class). We say that a category is *locally small* if every homesets is a set.

We proceed to introduce a comprehensive list of examples, so that it is already introduced in subsequent chapters.

Example 1.2.1.

- The elementary categories:
  - The category $0 = (\emptyset, \emptyset)$ where every property of metacategories is trivially satisfied.
  - The category $1 = (\{e\}, \{1_e\})$.
  - The category $2 = (\{a, b\}, \{1_a, 1_b, f : a \to b\})$
- Discrete categories: are categories where every arrow is an identity arrow. This are sets regarded as categories, in the following sense: every discrete category $C = (A, \{1_a : a \in A\})$ is fully identified by its set of object.
- Monoids and Groups: A monoid is a category with one object (regarding the monoid of the arrows). In the same way, if we requires the arrows to be invertible, we can see a group as a single-object category.
- Preorder: From a preorder $(A, \leq)$ we can define a category $C = (A, B)$ where B has an arrow $e : a \to b$ for every $a, b \in A$ such that $a \leq B$. The identity arrow is the arrow that arise from the reflexive property of the preorders.

[1]There is a discussion here that must be addressed at the end

- Large categories: these categories have a large set of objects. For example:
  - The category *Top* that has as objects all small topological spaces and as arrow continuous mappings.
  - The category *Set* that has as objects all small sets and as arrows all functions between them. We can also consider the category Set_* of pointed small sets (sets with a distinguish point), and functions between them that maps preserving the marked point. This category, when restricted to finite sets, is known as *FinSet*.
  - The category *Vect* That has as object all small vector spaces and as arrows all the linear functions.
  - The category *Grp* that has as object all small vector group and as arrows all the homomorphism.
  - The category Top_* that has as object all small topological spaces with a distinguished point, and as arrows all the continuous functions that maps each distinguished points into distinguished points. Similarly we can consider Set_* or Grp_*.
  - The category *Grph* that has as objects all small graphs, and as arrows all graphs morphisms.
- The category *Hask* of all Haskell types and all possible functions between two types.

Note that, for example, as natural numbers can be seen as either a set or a preorder, they also can be seen as a discrete category or a preorder category.

Simple enough categories can be easily described with graphs: there are as many objects as nodes in the diagram and an arrow in the category for each:

- Arrow in the graph.
- Composition of existing arrows.
- Every node (identity arrow usually omitted in diagrams).

For example, we can fully represent the 2 category with the diagram:

$$a \xrightarrow{g} c$$

1.2.1 Properties

We can see that is common in mathematics to have an object of study (propositional logic clauses, groups, Banach spaces or types in Haskell). Once the purpose of studying these particular sets of objects is fixed, it is also common to proceed to consider the transformations between these objects (partial truth assignments, homomorphisms, linear bounded functionals or functions in Haskell).

In categories, we have a kind of different approach to the subject. Instead of focusing on the objects themselves, we focus on how do they relate to each other. That is, we focus on the study of the arrows and how they composes. Therefore we can consider equal two objects that has the same relations with other objects. This inspire the next definition:

Definition 1.2.3. [29, Definition 1.1.9] Given a category $C = (O, A)$, a morphism $f : a \to b \in A$ is said to have a *left inverse* (resp. *right inverse*) if there exists a $g : b \to a \in A$ such that morphism $g \circ f = 1_b$ (resp. $f \circ g = 1_a$). A morpishm is an *isomorphism* if it has both left and right inverse, sloppily called the *inverse*. Two object are isomorphic is there exists an isomorphism between them.

Is easy to follow that if a morphism has a left and a right inverse, they must be the same, thus implying the uniqueness of the inverse. Also one can see that this functions define, by precomposition, bijections between $\hom(a, c)$ and $\hom(b, c)$ for all $c \in O$.

We will now proceed to talk about certain arrows and objects that have properties that distinguish them from others. The most useful example to gain an intuition of these properties is that of the $FinSet$ category, of all finite sets and functions between them. Considering special arrows:

Definition 1.2.4. An arrow f is *monic* (resp. *epic*) if it is *left-cancelable* (resp. *right-cancelable*), i.e. $f \circ g = f \circ h \implies g = h$ (resp. $g \circ f = h \circ f \implies g = h$).

In $FinSet$ these arrows are the injective functions (resp. surjective). Considering special objects:

Definition 1.2.5. an object a is *terminal* (resp. *initial*) if for every object b there exists an unique arrow $f : b \to a$ (resp. $f : a \to b$). An object that is both terminal and initial is called *zero*.

In $FinSet$ the initial object is the empty set and the terminal object the one point set.

Proposition 1.2.1. Every two terminal object are isomorphic.

Proof. Every terminal object has only one arrow from itself to itself, and necessarily this arrow has to be the identity. Let a, b be terminal object and $f : a \to b$ and $g : b \to a$ be the only arrows with that domain and codomain. Then $f \circ g : a \to a \implies f \circ g = 1_a$. Analogously $g \circ f = 1_b$.

□

Another important property is the *duality property*. This property tell us that for every theorem that we prove for categories, there exist another theorem that is automatically also true, by inverting the direction of the arrows. To formalize this idea, we define the concept of *opposite category*.

Definition 1.2.6. [29, Definition 1.2.1] Let C be any category. The opposite category C^{op} has:

- the same objects as C,
- an arrow $f^{op} : b \to a \in C^{op}$ for each arrow $f : a \to b \in C$, so that the domain of f^{op} is defined as the codomain of f and viceversa.

The remaining structure of the category C^{op} is given as follows:

- For each object a, the arrow 1_a^{op} serves as its identity in C^{op}.
- We observe that f^{op} and g^{op} are composable when f and g are, and define $f^{op} \circ g^{op} = (g \circ f)^{op}$.

The intuition is that we have the same category, only that all arrows are turned around. We can see that for each theorem T that we prove, we have reinterpretation that theorem to the opposite category. Intuitively this theorem is an equal theorem in which all the arrows have been turned around. For example: the proposition 1.2.1 can be reworked as:

Proposition 1.2.2. Every two initial object are isomorphic.

That is because being initial is the dual property of being terminal, that is, if $a \in C$ is a terminal object then $a \in C^{op}$ is an initial object. The property "being isomorphic" is its own dual.

1.2.2 Transformation in categories

One of the main ways of defining a category is considering as object every small set with an structure (e.g. groups, monoids, vector spaces), and as morphism all functions that preserve structure. Then, we may also follow our study defining the structure preserving transformation of categories.

Definition 1.2.7. Given two categories C, B, a *functor* $F : B \to C$ is a pair of functions $F = (F' : Ob(C) \to Ob(B), F'' : Ar(C) \to Ar(B))$ (the *object functor* and the *arrow functor* respectively) in such a way that:

$$F''(1_c) = 1_{F'c}, \forall c \in Ob(C), \qquad F''(f \circ g) = F''f \circ F''g, \ \forall f : a \to b, g : b \to c \in Ar(C).$$

Roughly speaking, a functor is a morphism of categories. When there is no ambiguity we will represent both F' and F'' with a single symbol F acting on both objects and arrows. Also, it can be seen in the definition that whenever possible the parentheses of the functor will be dropped. This loss of parentheses will be replicated throughout the text, whenever possible.

Remark 1.2.1. A functor F can be defined only pointing out how it maps arrows, as how F maps object can be defined with how it map identity arrows.

Lets provide some examples of functors:

Example 1.2.2.

- Forgetfull functor: We have a variety of categories consisting of structures with sets as objects and functions that hold structure such as arrows (eg Top, Grp or Vect).

  Let C one of such categories, then we have a functor $F : C \to Set$ that maps each object to its underlying set and each arrow to the equivalent arrow between sets. That is, this functor forgets about the structure that is present in C.

  Additionally, we will often have functors defined $F : C \to Set$. For some cases of C it also happen that the image of $Ob(C)$ by F' has some more structure on it. In that case we will say that F is an enriched functor for that cases.

  Another similar case of Forgetful functor is $\mathcal{U} : Cart \to Grph$ that maps each (small) category to their *underlying* graph.

- Fundamental group: In the context of algebraic topology we have the functor of the fundamental group $\Pi_1 : Top_* \to Grp$. The most famous property of this function is that

  > "Given continuous application $f, g : X \to Y$ between two topological spaces induces an application of the group of loops of X on the group of loops of Y such that $\Pi_1(f \circ g) = \Pi_1(f) \circ \Pi_1(g)$."

  that is, the functoriality of the function (taking also into account the fact that the identity of topological spaces is mapped to the identity of group).

- Stone-Čech compactification: From the realm of functional análisis we have the following theorem:

  Theorem 1.2.1. *Let $\mathbb{K} = \mathbb{R}$ or $\mathbb{C}$ and let Ω be a completely regular Hausdorff topological space. Then there exists a compact Hausdorff topological space $\beta\Omega$ and a homeomorphism Φ, from Ω to a dense subset of $\beta\Omega$. Moreover, for every continuous and bounded function $f : \Omega \to K$ there exists a unique $\overline{f} \in C(\beta\Omega)$ such that $\overline{f} \circ \Phi = f$ and the application $f \to \overline{f}$ is an isomorphism. Finally, $\beta\Omega$ is uniquely determined up to isomorphism by the above properties.*

  Proof. We define the functional $\Delta : \Omega \to \Omega^{**}$ such that $\Delta(t) = \delta_t$ where $\delta_t(f) = f(t)$ is Dirac's operator . It is enough to take as $\beta\Omega$ the closure of $\Delta(\Omega)$ in the induced weak-topology ω^* of Ω (since in the weak topology we have that a bounded closed is compact).

  Given a continuous and bounded function $f : \Omega \to \mathbb{K}$ we can define the function $\overline{f}$ by injection of Ω and then limit to maintain continuity (which will exist by compactness and boundedness of f). Thus the maximum of $\overline{f}$ can be reached by a sequence of points injected from Ω obtaining that

  $$\max \left\{ | \overline{f}\ (s) \mid : s \in \beta\Omega \right\} = \sup \{ |f(s)| : s \in \Omega \}.$$

  The application $f \mapsto \overline{f}$ is therefore an isometric isomorphism. Thanks to this isometric isomorphism the function spaces and by the Banach-Stone theorem[5, Theorem 3] we have uniqueness up to isomorphisms. □

  Based on this theorem we can define a functor $\beta : Hauss \to Comp$ where $Hauss$ is the category of Completely regular Haussdorff Spaces and continuous functions, and $Comp$ the category of Compact Haussdorff Spaces along with continuous function. This functor:

  - Take an object $\Omega \in Hauss$ to $\beta(\Omega) = \beta\Omega$:
  - Take an arrow $f : \Omega \to \Omega' \in Hauss$ to $\Delta(f)$ and the complete it by continuity as in the previous proof.

- Group actions: Every group action can be seen as a functor from a Group to a Set.

  Let G be a group seen as a category with only one element and X be a set seen as a discrete category. Then, an action is a representation of the group of the endomorphism of the set, that is, an action α associate each element of the group with an function $\alpha(g) : X \to X$ such that the product of element of the group is maintain via composition. That is for $g \in Ar(G)$ we have that $\alpha(g) \in Ar(X)$ and for all $g, f \in Ar(G)$:

$$\alpha(g \circ f) = \alpha(g) \circ \alpha(f).$$

Notice that, by the definition of G, the composition $g \circ f$ actually correspond to the product in G.

Note that is a group seen as a category the product is the composition thus having the functoriality.

- `Maybe` Method in Haskell [26, Section 7.1]: In Haskell the definition of Maybe is a mapping from a type a to a type Maybe a. We define the BN notation used in the example in A.1.

```
data Maybe a = Nothing | Just a
```

LISTING 1.1: Declaration of Maybe

Note that `Maybe a` is not a type but a function of types. In order for it to be an endofunctor of the *Hask* category we will need for it to also map function. Let $f : a \to b \in Hask$, then we define the function

$$\text{Maybe}\, f : \text{Maybe}\, a \to \text{Maybe}\, b$$

such that

$$(\text{Maybe}\, f)(Nothing) = Nothing, \qquad (\text{Maybe}\, f)(Just\ a) = Just\ f(a).$$

Note that $(\text{Maybe}\, id_a)(\text{Maybe}\, a) = Nothing \mid Just\ id_a(a) = \text{Maybe}\, a$.

We can now construct the category of all small categories *Cat*. This category has as object all small categories and as arrows all functor between them. Note that *Cat* does not contain itself, since it is a large category.

We can consider some properties in functors. Note that we can consider a functor $T : C \to D$ as a set function over homeset, that is, a function:

$$T : \hom_C(a, b) \to \{Tf : Ta \to Tb | f \in C\} \subset \hom_D(Ta, Tb)$$

Definition 1.2.8. A functor $T : C \to D$ is *full* if it is surjective as a function over homesets, i.e. if the function $T : \hom_C(a, b) \to \hom_D(Ta, Tb)$ is surjective for every $a, b \in C$. A functor is *faithfull* if it is injective over homesets.

As we have defined the concept of opposite category, we can consider functors $T : C^{op} \to D$. It is customary to refer to this type of functions a *contravariant* functor from C to D and regular functors $F : C \to D$ as *covariant*.

We shall continue defining natural transformation. In the words of Saunders Mac Lane:

> "Category has been defined in order to define functor and functor has been defined in order to define natural transformation."

One can see a functor $T : C \to D$ as a representation of a category in another, in the sense that a functor provide a picture of the category C in D. Further into this idea, we can consider how to transform these drawings into each other.

Definition 1.2.9. Given two functor $T, S : C \to D$, a *natural transformation* $\tau : T \Rightarrow S$ is a function from $Ob(C)$ to $Ar(D)$ such that for every arrow $f : c \to c' \in C$ the following diagram commutes:

$$\begin{tikzcd} Tc \arrow[r, "\tau c"] \arrow[d, "Tf"] & Sc \arrow[d, "Sf"] \\ Tc' \arrow[r, "\tau c'"] & Sc' \end{tikzcd} .$$

A natural transformation where every arrow τc is invertible is called a *natural equivalence* and the functors are *naturally isomorphic*.

Remark 1.2.2. In come context, subindex τ_c are used to note natural transformation, specially when working in composition with other arrows.

That is a natural transformation is a map from pictures of C in to pictures of D. Note that a natural transformation acts only on the domain of objects. Lets provide some examples.

When we have to define natural transformations it will often be useful to define each arrow individually. In this case, with the same notation as in the definition, we will note $\tau c = \tau_c : Tc \to Sc$.

Example 1.2.3.

- The opposite group: We can define the opposite functor $(\cdot)^{op} : Grp \to Grp$ that maps $(G, *)$ to $(G; *^{op})$ where $a *^{op} b = b * a$ and maps a morphism $f : G \to G'$ to $f^{op}(a) = f(a)$ and have that

  $$f^{op}(a *^{op} b) = f(b * a) = f(b) * f(a) = f^{op}(a) *^{op} f^{op}(b).$$

  Denoting the identity function $Id : Grp \to Grp$, we have a natural transformation $\tau : Id \Rightarrow (\cdot)^{op}$ defined by $\tau_G(a) = a^{-1}$ for all $a \in G$.

- In Haskell, the safeHead [26, Section 10.1] function between the List functor and the Maybe functor. To be precise,

  - The list functor maps the type a to the type $[a]$ and assigns each function $f : a \to b$ to $f' : [a] \to [b]$ that applies f elements wise (on empty list it does nothing).

    ```
    data List a = Nil | Cons a ( List a)
    ```

  - With this definition, we can define the natural transformation SafeHead : List $\to$ Maybe as:

    ```
    safeHead :: [a] -> Maybe a
    safeHead [] = Nothing
    safeHead (x : xs) = Just x
    ```

    or in category notation: $\tau : List \to Maybe$ such that, for every type a:

    $$\tau a([]) = \text{Nothing}, \qquad \tau a(x : xs) = \text{Just}\, x.$$

1.2.3 Constructions

In this last subsection we will introduce some standard construction on categories, along with some examples of these constructions.

Product Category

We present now one of the most usual construction in mathematics: the product. We will additionally consider the "universal" properties of product on the next chapter.

Definition 1.2.10. Let B, C be categories. Then the *product category* $B \times C$ is the category that has as objects the pairs $\{\langle b, c\rangle : b \in Ob(B), c \in Ob(C)\}$ and as arrows the pairs of arrows $\{\langle f, g\rangle : f \in Ar(B), g \in Ar(C)\}$. The composition of arrows is defined by the elementwise composition.

It is clear that we can define two functors $P : B \times C \to B$ and $Q : B \times C \to C$ that restricts the category to each of its component parts (functorial axioms follow immediately). Moreover, we can see that any functor $F : D \to B \times C$ will be uniquely identified by its composition with P and Q.

Complementary, for any two functors $F : D \to B, G : D \to C$ we can define a functor $F \times G : D \to B \times C$ that maps $(F \times G)\langle f, g\rangle = \langle Fg, Gg\rangle$. Expressed as a diagram:

$$\begin{tikzcd} & D \arrow[dl, "F"'] \arrow[d, dashed, "F\times G"] \arrow[dr, "G"] & \\ B & B \times C \arrow[l, "P"] \arrow[r, "Q"'] & C \end{tikzcd}$$

A functor $F : B^{op} \times C \to D$ is called a *bifunctor*. Arguably the most important bifunctor is the $\hom_C$ function. Given a category C we can see $\hom_C : C^{op} \times C \to Set$ as a bifunctor such that for all $a, b \in Ob(C)$:

$$\hom_C(\cdot, \cdot)(a, b) = \hom_C(a, b)$$

for the object. For the arrows, for all $f : a \to a', g : b \to b' \in C$ and for all $h \in \hom_C(a', b)$:

$$\hom_C(f^{op}, g) : \hom_C(a', b) \to \hom_C(a, b'),$$
$$\hom_C(f^{op}, g)(h) = g \circ h \circ f.$$

From this bifunctor we can define two functors for every $c \in Ob(C), g : d \to d' \in C, h \in \hom_C(c, d)$:

- The functor $\hom_C(c, \cdot) : C \to Set$ such that for all $d \in Ob(C)$:

$$\hom_C(c, \cdot) = \hom_C(c, d),$$

and

$$\hom_C(c, g) : \hom_C(a, d) \to \hom_C(a, d'),$$
$$\hom_C(c, g)(h) = g \circ h.$$

This functor is often called the *post-composition* functor.

- The covariant functor $\hom_C(\cdot, c) : C^{op} \to Set$, defined analogously.This functor is often called the *pre-composition* functor.

Functor Categories

We continue by defining functor categories, that is, categories where we consider the functors as objects and natural transformation as arrows in some sense. This concept will be instrumental in further consideration in the realm of functional programming (in particular, in the definition of a monad).

Before explaining composition between natural transformation, it is useful to present the following diagram. Let B, C be categories, $F, G : B \to C$ be functors and $\tau : F \to G$ natural transformation τ. It is common to represent this structure with:

$$\begin{tikzcd} A \arrow[r, bend left=50, "F"{name=U}] \arrow[r, bend right=50, "G"'{name=D}] & B \arrow[from=U, to=D, Rightarrow, "\tau"] \end{tikzcd}$$

Let us define the composition of two natural transformation. We will start defining the composition of two natural tranformations σ, τ as:

$$\begin{tikzcd} A \arrow[r, bend left=70, "F"{name=U}] \arrow[r, "G"{name=M}] \arrow[r, bend right=70, "H"'{name=D}] & B \arrow[from=U, to=M, Rightarrow, "\tau"'] \arrow[from=M, to=D, Rightarrow, "\sigma"'] \end{tikzcd}$$

Due to this representation, this composition of natural transformations is called *vertical composition*, in opposition to the *horizontal composition* (def. 1.2.13).

Definition 1.2.11. Let C and B be two categories, $R, S, T : C \to B$ be functors, and let $\tau : R \to S, \sigma : S \to T$. We define the composition $(\tau \circ \sigma)c = \tau c \circ \sigma c$.

To see that $(\tau \circ \sigma)$ is a natural transformation it suffices the following diagram [34]:

$$\begin{tikzcd} Rc \arrow[r, "Rf"] \arrow[d, "\sigma c"'] \arrow[dd, bend right=60] & Rc' \arrow[d, "\sigma c'"] \arrow[dd, bend left=60] \\ Sc \arrow[r, "Sf"] \arrow[d, "\tau c"'] & S(c') \arrow[d, "\tau c'"] \\ Tc \arrow[r, "Tf"] & Tc' \end{tikzcd}$$

Definition 1.2.12. Let B, C be categories. We define the Functor category from C to B as B^C as the category with all functors $F : C \to B$ as object, natural transformation as arrows, and composition as defined in 1.2.11.

We now present some examples to provide some intuition about when this type of construction.

Example 1.2.4.

- The category of group actions over sets: As we have seen in example 1.2.2 each group action over a set is a functor. Let G be a group and S be a set, both seen as categories and discrete categories. Then we can consider the category S^G, that has as object every group action of G over S and as arrow the morpishm of actions.

- In example 1.2.1 we define the 2 category and can consider therefore the category of functor from 2 to C. This category is called the arrow category, as the can see that there are a functor from 2 to C as functor for each arrow in C and conversely.

  This example displays a interesting idea: we can consider small collection of object as functions/functors. This idea can as well be seen when we define a pair of real numbers as any function $f : \{0,1\} \to \mathbb{R}$. Should we want, for example, to consider all the squares in a category C, we might ass well define the category square Sqr and consider every functor $F : Sqr \to C$.

$$\begin{tikzcd} a \arrow[r, "f"] \arrow[d, "g"] & a' \arrow[d, "h"] \\ b \arrow[r, "k"] & b' \end{tikzcd}$$

  The square category.

  Note that the elements $F : Sqr \to C$ are the collections of the arrows of the arrow category of C.

- We can use this construction to study the category C^C of endofunctors of a particular category C. This case is particularly interesting while studying the endofunctors present in $Hask^{Hask}$ and later consider the Monad as a programming structure.

Note that this is not the only way in which to define the composition of natural transformations can be defined. In fact, we can define another functor category. In this case we compose two natural transformation as in:

$$\begin{tikzcd} B \arrow[r, bend left, "F"] \arrow[r, bend right, "G"'] \arrow[r, phantom, "\Downarrow \tau"] & C \arrow[r, bend left, "F'"] \arrow[r, bend right, "G'"'] \arrow[r, phantom, "\Downarrow \sigma"] & D \end{tikzcd}$$

Lets formalize this composition:

Definition 1.2.13. Let B, C, D be categories, $F, G : B \to C, F', G' : C \to D$ be functors, and let $\tau : F \to G, \sigma : F' \to G'$, we define the horizontal composition $(\tau \circ \sigma) : F \circ F' \to G \circ G'$ by:

$$(\sigma \circ \tau)c = \sigma Fc \circ G'\tau c$$

for all $c \in B$.

In this case we can see that the composition of two natural transformation is indeed a natural transformation due to the commutativity of:

$$\begin{tikzcd}
F'Fc \arrow[r, "\sigma Fc"] \arrow[d, "F'\tau c"] \arrow[dr, "(\sigma\circ\tau)c"] & G'Fc \arrow[d, "G'\tau c"] \\
F'Gc \arrow[r, "\sigma Gc'"] & G'Gc'
\end{tikzcd}$$

With this definition of composition we can consider another different category: the category of all functors of all (small) categories, that is, the category that has all the functors as object, and has the natural transformation with horizontal composition as arrows.

When we have to consider both compositions at the same time we denote the vertical composition with $\tau \cdot \sigma$ and horizontal composition with $\tau \circ \sigma$, as in [24]. Lastly we have to consider how this composition relate to each other. This is seen in the *interchange law*:

Proposition 1.2.3. Let A, B, C be categories, let $F, G, H : A \to B$ and $F', G', H' : B \to C$ be functors and $\tau : F \to G, \sigma : G \to H, \tau' : F' \to G'$ and $\sigma' : G' \to H'$ be natural transformations. Then:

$$(\sigma' \circ \sigma) \cdot (\tau' \circ \tau) = (\sigma' \cdot \tau') \circ (\sigma \cdot \tau)$$

Proof. We have a structure like

$$\begin{tikzcd}
A \arrow[r, bend left=60, "F"] \arrow[r, "G"] \arrow[r, bend right=60, "H"'] \arrow[r, phantom, shift left=4ex, "\tau\Downarrow"] \arrow[r, phantom, shift right=4ex, "\sigma\Downarrow"] & B \arrow[r, bend left=60, "F'"] \arrow[r, "G'"] \arrow[r, bend right=60, "H'"'] \arrow[r, phantom, shift left=4ex, "\tau'\Downarrow"] \arrow[r, phantom, shift right=4ex, "\sigma'\Downarrow"] & C
\end{tikzcd}$$

From the naturality of τ' we have that for all $c \in A$:

$$\begin{aligned}
((\sigma' \cdot \tau') \circ (\sigma \cdot \tau))(c) &= H'((\sigma \cdot \tau)c) \circ (\sigma' \cdot \tau')(Fc) \\
&= H'(\sigma c \circ \tau c) \circ (\sigma'(Fc) \circ \tau'(Fc)) \\
&= H'(\sigma c) \circ H'(\tau c) \circ \sigma'(Fc) \circ \tau'(Fc) \\
&= H'(\sigma c) \circ \sigma' Gc \circ G'\tau c H'(\tau c) \circ \sigma'(Fc) \circ \tau'(Fc) \\
&= (\sigma' \circ \sigma)(c) \circ (\tau \circ \tau)(c) \\
&= ((\sigma' \circ \sigma) \cdot (\tau' \circ \tau))(c).
\end{aligned}$$

□

To finally consider the relation between products, and the functor category we can see that given three small categories A, B, C we have a bijection:

$$\hom_{Cat}(A \times B, C) \cong \hom_{Cat}(A, C^B).$$

This fact will be taken into account further in the text, as this will mean that the functor $\cdot \times B : Cat \to Cat$ has a right adjoint.

Comma Category

To define the comma category, we shall define the category $(b \downarrow S)$ of objects S-under b, sketch the dual notion, and generalize this two concepts with the comma category.

Given a functor $F : B \to C$ an object $b \in B$ is $F-$under another object $c \in C$ if there exists an arrow $f : c \to b \in C$. This can be represented as

$$\begin{tikzcd} c \arrow[d, "f"] \\ Fb \end{tikzcd}$$

and thus the name of F-under.

Definition 1.2.14. Let B, C be categories and $S : B \to C$ be a functor. For every $c \in Ob(C)$ we can define the category $(c \downarrow S)$ that has as objects all pairs $(b, f) \in Ob(B) \times Ar(C)$ where $f : c \to Sb$, and as arrows $h : (d, f) \to (d', f')$ the arrows $h : d \to d' \in Ar(B)$ such that $f' = Sh \circ f$.

The property that each arrow should satisfy can be represented as:

$$\begin{tikzcd} & c \arrow[dl, "f"'] \arrow[dr, "f'"] & \\ Sd \arrow[rr, "Sh"'] & & Sd' \end{tikzcd}$$

Example 1.2.5. Let $U : Grp \to Set$ be the forgetful functor, and let $X \in Ob(Set)$. We can consider $(X \downarrow U)$ where every object is a function of Sets $f : X \to Ug$ for a group g.

One can easily now deduce the dual concept of the category $(b \uparrow S)$ represented by:

$$\text{objects: } \begin{tikzcd} c \\ Fb \arrow[u, "f"] \end{tikzcd} ; \qquad \text{arrows: } \begin{tikzcd} & c & \\ Sd \arrow[ur, "f"] \arrow[rr, "Sh"'] & & Sd' \arrow[ul, "f'"'] \end{tikzcd} .$$

Now suppose that we have three categories B, C, D and two functors S, T such that

$$B \xrightarrow{S} C \xleftarrow{T} D$$

We might want to consider the relations objects of B and D, for that we have the comma category:

Definition 1.2.15. Let B, C, D be categories and to $S : B \to C, T : C \to D$. We define the coma category $(S \downarrow T)$ as the category that has as object the triples (b, d, f) with $b \in Ob(B), d \in Ob(D), f : Sb \to Td \in Ar(C)$ and as arrows the pairs $(g, h) : (b, d, f) \to (b', d', f')$ with $g : b \to b' \in Ar(B), h : d \to d' \in Ar(D)$ such that $Th \circ f = f' \circ Sg$.

We can represent the previous definition by:

$$\text{objects: } \begin{tikzcd} Sb \arrow[d, "f"] \\ Td \end{tikzcd} ; \qquad \text{arrows: } \begin{tikzcd} Sb \arrow[r, "Sg"] \arrow[d, "f"] & Sb' \arrow[d, "f'"] \\ Td \arrow[r, "Sh"] & Td' \end{tikzcd} .$$

The name comma category comes from is alternative notation $(S \downarrow T) = (S, T)$. We prefer the $(S \downarrow T)$ as it is more clear, nonetheless it is clear that before modern text editor became popular, the original comma notation had a big plus.

Chapter 2

Universality, Adjoints and Closed Cartesian Categories

In this chapter we are going to study universality and adjunctions. Despite a possible apparent difference in their definitions, both have the same use: they are tools that, given an arrow selected in a category, allow us to uniquely select an arrow in another category, generally fulfilling some special property.

We will start this section with the definition of universality, as well as the statement of the celebrated Yoneda's lemma. Then, we present the concept of adjunction. We will use these concepts to introduce new structures, in particular that of closed Cartesian category.

2.1 Universality

In this section we present the concept of universality. This concept is behind lots of mathematical properties. Intuitively, universality is an efficient way of expressing a one-to-one correspondence between arrows of different categories. This one-to-one relationship is usually expressed via "given an arrow f it exists one and only one arrow $\overline{f}$ such that <insert your favorite universal property>".

Probably the first contact that any mathematician has with universality is defining a function $f : \mathbb{R} \to \mathbb{R}^2$. It is easy to see that defining such a function is equivalent to define two $g, h : \mathbb{R} \to \mathbb{R}$. Furthermore, those g, h are unique for each f. This uniqueness is the flavor that attempts to capture the concept of universality. Other examples of unique existence are those that occur in quotient groups or in bases of vector spaces.

Definition 2.1.1. Let $S : D \to C$ be a functor and $c \in Ob(C)$. An *universal arrow* from c to S is a pair (d, u) with $d \in Ob(D), u : c \to Sd \in Ar(C)$, such that for every (e, f) with $e \in Ob(D)$ and $f : c \to Sd$ there exists an unique $f' : r, d \in Ar(D)$ such that $Sf' \circ u = f$.

In a diagram:

$$\begin{tikzcd} c \arrow[r, "u"] \arrow[dr, "f"'] & Sd \arrow[d, dashed, "Sf'"] & d \arrow[d, dash, "f'"] \\ & Se & e \end{tikzcd}$$

Note that an universal arrow (d, u) induces the unique existence of an arrow in D, but with interesting properties via it relationship with S. Usually, to provide an universal arrow we will only define the functor $S : D \to C$ and the arrow $u : c \to Sd$, letting all other information be deduced from the context.

The idea of universality is often regard via the notion of *universal element*, in the special (and commonplace) case where $S : D \to Set$.

Definition 2.1.2. Let $S : D \to Set$, an *universal element* for S is a pair (r, e) with $r \in Ob(D), e \in Sr$, such that for every (d, x) with $d \in Ob(D), x \in Sd$, there is an unique arrow $f : r \to d \in D$ such that $(Sf)e = x$.

Note that having an universal element is a particular case of having an universal arrow. Considering $*$ the set with one point (as a category) an universal element is an universal arrow $* \to H$. This is clearly seen if we consider the diagram:

$$\begin{tikzcd}
* \arrow[r, "u"] \arrow[dr, "f"'] & Sr \arrow[d, dashed, "Sf"] & r \arrow[d, dash, "f"] \\
& Sd & d
\end{tikzcd}$$

where $*$ is helping us select the elements of Sd and Se to enforce the property.

Conversely if D has small homesets, we can consider an universal arrow to be a particular case of an universal element. With the same context as in 2.1.1 $(d, u : c \to Sd)$ is an universal arrow if, and only if, $(d, u \in \hom_D(c, Sd))$ is an universal element.

The point of having these such similar definition is to have different interfaces for the concept. While most result can be easily adapted, once we got to the examples is clear when to use each concept.

Example 2.1.1.

- Quotient group: This property states that, for any two groups $N \lhd G$ with $\pi : G \to N$ the canonical projection, and any group homomorphism $f : G \to K$ such that $N \subset \ker f$ there exists an unique $\overline{f} : H \to K$ such that $\overline{f} \circ \pi = f$.

  The question now is, how is this a universal property. This is an example of a universal element. Considering the functor $H : Grp \to Set$:

  $$HG' = \{f : G \to G' \in Ar(Grp) : F(N) = \{0\}\},$$

  Then, $(G/N, \pi)$ is an universal element for H. From this property alone the three isomorphism theorems can be deduced. Therefore, we only have to prove this result to have the full power of these theorems in any context (e.g. Rings, K-Algebra, or Topological spaces).

- Tensor product: Given an right R-module A and a left R-module B one can consider the tensor product as an abelian group $A \otimes_R B$ and an R-biadditive function

  $$h : A \times B \to A \otimes_R B,$$

  such that for every R-biadditive $f : A \times B \to G$ it exists an unique $\overline{f}$ such that $\overline{f} \circ h = f$. Is easy to follow that tensor product can be reformulated as an universal arrow $\varphi : AB \to A \otimes_R B$ from $A \times B$ to the identity functor in the category of R-additive groups along with R-additive functions.

Dually, we can consider an *universal arrow from S to c* or simply universal arrow from S:

Definition 2.1.3. Let $S : D \to C$ be a functor and $c \in Ob(C)$, an *universal arrow* from c to S is a pair (d, u) with $d \in Ob(D), u : c \to Sd \in Ar(C)$, such that for every (e, f) with $e \in Ob(D)$ and $f : c \to Sd$ there exists an unique $f' : r, d \in Ar(D)$ such that $u \circ Sf' = f$.

In a diagram:

$$\begin{tikzcd} c & Sd \arrow[l, "f"'] \arrow[d, dashed, "Sf'"] & d \arrow[d, "f'"] \\ & Se \arrow[ul, "u"] & e \end{tikzcd}$$

Example 2.1.2.

- In *Set* we have a particular construction: the product of sets for any two sets a, b along with the projections $\pi_1 a \times b \to a$ and $\pi_2 a \times b \to b$. Then, we can define the functor $\Delta : Set \to Set \times Set$ such that $\Delta c = c \times c$ for every $c \in Ob(Set)$ and $\Delta(f : c \to b) : c \times c \to b \times b$ is the elements-wise application of f.

  Then, given any two sets $a, b \in Ob(Sets)$, $u = (\pi_1, \pi_2)$ is an universal arrow from Δ to (a, b) as

  $$\begin{tikzcd} (a,b) & \Delta d = (d,d) \arrow[l, "f"'] \arrow[d, dashed, "\Delta f'"] & d \arrow[d, "f'"] \\ & \Delta e = (e,e) \arrow[ul, "u"] & e \end{tikzcd}$$

  This construction is reproducible in, among many others, Grp, Top or Banach Spaces with bounded linear transformation.

Lastly, we will provide a characterization of universality:

Proposition 2.1.1. Let $S : D \to C$ be a functor and $u : c \to Sr \in Ar(C)$. Then u is an universal arrow if and, only if, the function $\varphi : \hom_D(r, \cdot) \to \hom(c, S\cdot)$ such that $\varphi(d)(f) = Sf \circ u$ for all $f \in \hom_D(r, d)$ is a natural bijection. Conversely, every natural bijection is uniquely determined by an universal arrow $u : c \to Sr$.

Proof. Let u be universal. Then, φ is clearly a bijection. For φ to be a natural transformation the diagram

$$\begin{tikzcd} \hom_D(r,d) \arrow[r, "\varphi(d)"] \arrow[d, "\hom_D(r,g)"'] & \hom_C(c,Sd) \arrow[d, "\hom_C(c,Sg)"] \\ \hom_D(r,d') \arrow[r, "\varphi(d')"] & \hom_C(c,Sd') \end{tikzcd}$$

should commute for all $g \in Ar(D)$. As this is a diagram in the category of sets, we can check the commutativity by checking it element wise. For any $f \in \hom_D(r,d)$:

$$\begin{tikzcd} f \arrow[r, "\varphi(d)"] \arrow[d, "\hom_D(r{,}g)"'] & Sf \circ u \arrow[d, "\hom_C(c{,}Sg)"] \\ g \circ f \arrow[r, "\varphi(d')"] & S(g \circ f) \circ u = Sg \circ Sf \circ u \end{tikzcd}$$

So the diagram commutes, and φ is natural. The bijectivity follows from u being universal.

Lets consider now that φ is a natural bijection. We will define $u := \varphi(r)(1_r)$ and check that (r, u) is an universal arrow. As φ is natural we have that:

$$\begin{tikzcd} \hom_D(r,r) \arrow[r, "\varphi(r)"] \arrow[d, "\hom_D(r{,}g)"'] & \hom_C(r, Sr) \arrow[d, "\hom_C(r{,}Sg)"] \\ \hom_D(r,d) \arrow[r, "\varphi(d)"] & \hom_C(r, Sd) \end{tikzcd}$$

Writing the diagram for the element $1_r \in \hom_D(r,r)$ and any $d \in Ob(D), g : r \to d \in \hom_D(r,d)$:

$$\begin{tikzcd} 1_r \arrow[r, "\varphi(r)"] \arrow[d, "\hom_D(r{,}g)"'] & u \arrow[d, "\hom_C(r{,}Sg)"] \\ g \arrow[r, "\varphi(d)"] & \varphi(d)(g) = Sg \circ u \end{tikzcd}$$

and since φ is a bijection, for every $f \in \hom_C(r, Sd)$ there is an unique function $f' = \varphi(d)^{-1}(f)$ such that $Sf' \circ u = f$, thus being u universal.

□

This proposition, sometimes called Yoneda's proposition [24, p. 81], is of capital importance. Later on this chapter, we will study *adjunctions* and this proposition will enable us to fully understand the relationship between adjunctions and universality. From result there is a definition that arises:

Definition 2.1.4. Let D be a category with small home-sets and let $F : D \to C$ be a functor. A representation of a functor $K : D \to Set$ is a pair (r, φ) with $r \in Ob(D)$ and φ a natural isomorphism such that

$$D(r, \cdot) \equiv_\varphi F \cdot .$$

A functor is said to be *representable* whenever it has a representation.

Note that therefore a universal arrow induces a natural isomorphism $D(r,d) \equiv C(c, Sd)$ and this induces a representation of the functor $C(c, S\cdot) : D \to Set$, being these three equivalent.

2.1.1 Yoneda's lemma

This subsection deals with Yoneda's Lemma. Mac Lane[24] assures the lemma first appeared in his private communication with Yoneda in 1954. With time, this result has became one of the most relevant one in Category Spaces. We will start by providing some intuition to it, followed by it proof and some use cases.

This results is due to Japanese professor Nobuo Yoneda. We know about Yoneda's life thanks to the elegy that was written by Yoshiki Kinoshita[38]. Yoneda was born in Japan in 1930, and received his doctorate in mathematics from Tokyo University in 1952. He was a reviewer for international mathematical journals. In addition to his contributions to the field of mathematics, he also devoted his research to computer science.

The idea behind the Yoneda lemma can be arid at first, if one does not have a prior understanding of what the purpose and usefulness of this lemma is. In order to illustrate this idea we will introduce a (simplified) definition of Moduli Spaces, so that we have a geometric understanding of Yoneda's Lemma.

The idea behind (some) Moduli spaces is to classify algebraic curves up to isomorphisms. In addition, Moduli spaces allow us to control complex mathematical objects (such as a quotient space of unknown objects) by simpler objects or objects with better properties (such as a concrete variety). A canonical example of this type of classification is:

$$\begin{aligned} \{\text{Vector spaces of finite dimension}\}/\text{isomorphism} &\cong \mathbb{N} \\ [V] &\to dimV. \end{aligned}$$

Where a complex object can be classified by another object of which we know more properties. We can start by defining:

$$\mathcal{M} = \{\text{smooth complex non singular curves}\}/\text{isomorphism}$$

Further on when we talk about curves we will refer to smooth complex non singular curves. Note that if two curves are isomorphic then they have the same genus. Therefore the function

$$\begin{aligned} \gamma : \mathcal{M} &\mapsto \mathbb{N} \\ [V] &\mapsto \text{genus of V} \end{aligned}$$

is well defined and we can define $\mathcal{M}_g = \gamma^{-1}(g)$. An interesting classification of $\mathcal{M}_g$ is given when we consider that for every g there exists a closed, connected, non-singular variety U_g and a family $\{C_t : t \in U_g$ such that a curve of genus g will be a fibration of C_t. Moreover there is a variety M_g and a subjective morphism $\varphi : U_g \to M_g$ such that $\varphi(t_1) = \varphi(t_2)$ if $C_{t_1} \equiv C_{t_2}$. Therefore we are classifying the equivalence classes of $\mathcal{M}_g$ by points of the variety M_g (thus generating a Moduli problem).

Similarly to this two example, with the Yoneda lemma we will have a functor $F : D \to Set$ and one representation of this functor. We will classify the natural transformation of these functors by the set in the image of F! Interestingly enough, there will be applications where the complex object is not the space of naturals transformations, but the images of F (see 2.1.2 for an example). Lets proceed to enunciate and proof the result.

Theorem 2.1.1. *[24, Section 3.2] Let D be a category with small home-sets, $K : D \to Set$ be a functor, and $r \in Ob(D)$. Then there is a bijection*

$$\tau : Nat(\hom_D(r,\cdot), K\cdot) \equiv Kr$$
$$\tau(\alpha : \hom_D(r,\cdot) \to K\cdot) = \alpha(r)(1_r)$$

Where τ is natural in K(as an object of Set^D) and in r.

Proof. As α is a natural transformation we have that, for every $g : r \to d \in Ar(D)$:

$$\begin{tikzcd}
\hom_D(r,r) \arrow[r, "\alpha(r)"] \arrow[d, "\hom_D(r,g)"'] & Kr \arrow[d, "Kg"] \\
\hom_D(r,d) \arrow[r, "\alpha(d)"] & Kd
\end{tikzcd}$$

Writing $\alpha(r)(1_r) = u$ we have that:

$$\begin{tikzcd}
1_r \arrow[r, "\alpha(r)"] \arrow[d, "\hom_D(r,g)"'] & u \arrow[d, "Kg"] \\
g \circ 1_r = g \arrow[r, "\alpha(d)"] & \alpha(d)(g) = Kg \circ u
\end{tikzcd}$$

Therefore every natural transformation is uniquely identified by the value of u, therefore τ is injective. Moreover, for every u in Kr, we can define a natural transformation following the previous diagram, therefore, τ is bijective.

To see that τ we is natural we have to consider for which functor it is natural. Consider the functor *evaluation* $E : Set^D \times D$ that maps each $(F, c) \to Fc$, and the functor $N : Set^D \times D \to Nat(\hom_D(r,\cdot), K)$ the set of natural transformations. Finally, $\tau : N \to E$ is a natural transformation. □

Remark 2.1.1. The existence of an evaluation will be revisited when we consider *closed cartesian cateogries* in 2.2.2. This categories (in general, closed categories) will generalized the idea of a category having an exponential object suitable for evaluation. In this case, we are unadvertedly using that *Cat* is a closed cartesian category.

The first functor that we will want to apply this result is to the $\hom_{\cdot}$ functor. But this functor is a bifunctor, so to get the full result of this lemma applied to the full bifunctor we may restate this lemma to contravariant functors.

Corollary 2.1.1.1. *Let D be a category with small home-sets, $F : D \to Set$ be a contravariant functor, and $r \in Ob(D)$. Then there is a bijection*

$$\tau : Nat(\hom_D(\cdot, r), F\cdot) \equiv Fr$$
$$\tau(\alpha : \hom_D(\cdot, r) \to F\cdot) = \alpha(r)(1_r)$$

Where τ is natural in K(as an object of Set^D) and in r.

Proof. We seek to use the Yoneda lemma in the functor $F' : D^{op} \to Set$ induced by F. Then we have that:

$$\tau : Nat(\hom_{D^{op}}(r,\cdot), F'\cdot) \equiv F'r$$
$$\tau(\alpha : \hom_{D^{op}}(r,\cdot) \to F'\cdot) = \alpha(r)(1_r)$$

Taking into account that $F_{|Ob(D)} = F'_{|Ob(D^{op})}$, and that $\hom_{D^{op}}(r, \cdot) = \hom_D(\cdot, r)$ we have the result. □

As we have seen, the Yoneda lemma is a direct generalization of the moduli problem. In the same vein, Yoneda's lemma is the generalisation of other problems/theorems in mathematics, most notably Cayley's lemma. It states:

Proposition 2.1.2. Any group is isomorphic to a subgroup of a symmetric group.

To understand this, take a groups G seen as a single-object category, and name that object e. Then, the functor $\hom_G(e, \cdot) : G \to Set$ can be seen as a group action 1.2.2. Then the Yoneda lemma states that:

$$Nat(\hom_G(e, \cdot), Hom_G(e, \cdot)) \equiv_\varphi Hom_G(e, e).$$

Translating this result to group theory:

- Remember that $Hom_G(e, e)$ is the group G.
- Every natural transformation is a equivariant map between G-sets.
- This equivariant maps, forms an endormophism group under composition, being a subgroup of the group of permutations.
- This natural isomorphism φ define a group isomorphism.

So we have the isomorphism of groups that is stated in Cayley's Theorem.

We continue our exploration of Yoneda lemma by defining the *Yoneda Embedding*. For that we define the contravariant functor $h_a = \hom_C(\cdot, a)$. Then the contravariant Yoneda lemma tell us that:

$$Nat(h_a, h_b) \equiv_{\tau_a} \hom(a, b).$$

We then can define a fully faithfuls embedding $v : C \to Set^{C^{op}}$ such that

$$\begin{aligned} va &= \hom_C(A, \cdot) && \forall a \in Ob(C), \\ vf &= \tau_a^{-1}(f) && \forall f : b \to a \in Ar(C). \end{aligned}$$

This functor allows us to view the category C as a subcategory of the category of contravariant functors from C to Set, which will be useful for determining "heritable" properties in C.

2.1.2 Properties expressed in terms of universality

After the examples given, we define a few constructions that are commonplace in Maths. We will outline the notions of limit, pullback and product, and the dual notions of colimit, pushout and coproduct.

The notions of product and pullback can be seen as particular cases of the notion of limit. To define limit we will introduce the concept of co-cone and the diagonal functor.

Definition 2.1.5. Let C, J be categories. We can define the functor $\Delta_J : C \to C^J$ that maps c to the functor from J to C that is constantly c, and maps every arrow to the identity 1_c.

Whenever possible we will write only Δ, and let the information of the category be deduced from context. J is usually small and often finite. We can now consider a natural transformation $\tau : F \to \Delta c$. This can be represented as in the following diagram:

$$\begin{tikzcd}
Fx_j \arrow[rr, "Fg"] \arrow[dr, "\tau x_j"'] & & Fx_k \arrow[dl, "\tau x_k"] \\
& c &
\end{tikzcd}$$

commutes for every $g : x_j \to x_k \in Ar(j)$, for that reason, such natural transformation is usually called a co-cone. The dual notion is called cone and is represented as:

$$\begin{tikzcd}
Fx_j \arrow[rr, "Fg"] & & Fx_k \\
& c \arrow[ul, "\tau x_j"] \arrow[ur, "\tau x_k"'] &
\end{tikzcd}$$

We can now define the concepts of limit and colimit. We introduce first the concept of colimit. This definition is that of an universal arrow, only in a category of functors. Following this definition, we will define the limit as its dual concept.

Definition 2.1.6. A colimit is an object $r \in Ob(C)$ together with an universal arrow $u : F \to \Delta r \in Ar(C^J)$. The colimit is denoted by

$$\lim_{\leftarrow} F = r = \operatorname{colim} F.$$

The notation $\lim_{\leftarrow}$ is intuitive: in the colimit we have arrows to F. To represent this as a diagram, we have a co-cone $u \to \operatorname{colim} F$ such that for every other co-cone $\tau \to s$, it exist an unique f such that the following commutes for every $x_j, x_k \in Ob(C)$:

$$\begin{tikzcd}
& l & \\
& \operatorname{colim} F \arrow[u, dashed, "f"] & \\
x_j \arrow[rr, "Fg"'] \arrow[ur, "ux_j"] \arrow[uur, bend left, "\tau x_j"] & & x_k \arrow[ul, "ux_k"'] \arrow[uul, bend right, "\tau x_k"']
\end{tikzcd}$$

Now, thanks to the duality of categories, we can define what a limit is in a very synthetic way:

Definition 2.1.7. A limit is the dual concept of a colimit. It is denoted as

$$\lim_{\rightarrow} F = r = \lim F.$$

A limit is represented by the following diagram:

$$\begin{tikzcd}
 & l \arrow[d, dashed, "f"] \arrow[ddl, bend right, "\tau x_j"'] \arrow[ddr, bend left, "\tau x_j"] & \\
 & \lim F \arrow[dl, "u x_j"'] \arrow[dr, "u x_k"] & \\
x_j \arrow[rr, "Fg"'] & & x_k
\end{tikzcd}$$

Analogously as in the colimit notation, in the limit we have arrows from F, and thus the notation $\lim_{\rightarrow}$. Let focus for a while now in the notion of limit, in particular of two of its specials cases: the product and the pullback. From these cases we are going to provide most examples.

We have already talked about the product of categories. The product is a limit where J is the 2-element discrete category, that is, where every functor $F : J \to C$ is merely choosing to object of C.

Definition 2.1.8. Let C be a category, $J = \{0, 1\}$ be the discrete category with two elements. The product $c_1 \times c_2$ of two elements $c_0, c_1 \in Ob(C)$ is the limit of the functor $F : J \to C$ such that $F0 = c_0, F1 = c_1$.

This construction means that providing an arrow to c_0, c_1 determines an unique arrow to $c_0 \times c_1$ and vice versa by composition with π_i. In this case, the arrows $u0, u1$ are usually called *projections* and denoted by π_0, π_1. The notation is due to the product being a generalization of the Cartesian product in the category *Set*. It is also sometimes note with $c_0 \Pi c_1$ with Π being standard for the sequence product. Some examples of product object in categories are:

Example 2.1.3.

- Products from Example 2.1.2.
- In Haskell, considering two integer types `A` and `B`, we can consider the duple type `(A,B)` that is the product, along with projections `fst` and `snd` to be a product. We can fairly easily check that, for any other integer type `C` and any morphism `f: C→ (A,B)` it exist a unique `g` and `h` such that:

$$\begin{tikzcd}
 & C \arrow[dl, dashed, bend right, "g"'] \arrow[d, dashed, "f"] \arrow[dr, dashed, bend left, "h"] & \\
A & (A, B) \arrow[l, "fst"'] \arrow[r, "snd"] & B
\end{tikzcd}$$

Note that we have requested `A` and `B` to be integer types. Quite surprisingly, this construction can not be generalized in Haskell to provide a product type for every object[15]. This will provide in the future a palpable difference between our theoretical λ-calculus, and its applicable version, Haskell.

This troubles arise from of implementation details of bottom values. Why have not anyone fix this? Well, when functions have to terminate and only finite values are considered this problems does not Arise. This condition is fairly acceptable in any engineering scenario.

Analogously, we can define the coproduct, on which instead of defining an arrow to c_0, c_1, we define an object *from* c_0, c_1.

Definition 2.1.9. The coproduct is the dual definition of the product. It is denoted by $c_0 \sqcup c_1$.

In this notation $\sqcup$ denotes an inverted Π, with the meaning of being the dual notion of the product.

Example 2.1.4.

- In the category Ab of Abelian Group and morphism, finite products and coproducts are equivalent. This equivalence also happens in others categories, such as R-Modules for a ring R or vector spaces. This has a underlying idea of a type of category that relates this constructions: This are all *Abelian Categories*, as in [30, Section 5.5].

  We will not delve further into this concept so as not to deviate from the subject at hand. However, for those readers who are familiar with the notation, we cannot resist including Mitchel's theorem[27, Chapter IV] .

  Theorem 2.1.2 (Mitchell's Theorem)**.** *If A is a small abelian category, then there is a covariant full faithful exact functor $F : A \to Ab$.*

  As a consequence, any statement "p implies q" where p and q are categorical statement about a diagram in A, if is true in Ab, it holds true in A.

After learning about the product and the coproduct, we will focus now in the notion of pullback and its dual, the pushout. We will first define the category

$$P = \quad x \xrightarrow{f} z \xleftarrow{g} y$$

Then we can define the pullback:

Definition 2.1.10. Let C be a category and $F : P \to C$ be a functor. Then the pullback of Fx and F_y denoted as $Fx \times_{Fz} Fy$ is the limit of the functor F.

We can represent this structure in the following diagram. For any object q and arrows $f' : q \to Fx, g' : q \to Fy$ we have:

$$\begin{tikzcd}
q \arrow[dr, dashed, "u"] \arrow[drr, bend left, "g'"] \arrow[ddr, bend right, "f'"'] & & \\
& Fx \times_{Fz} Fy \arrow[r, "p_2"] \arrow[d, "p_1"'] & Fy \arrow[d, "Fg"] \\
& Fx \arrow[r, "Ff"'] & Fz
\end{tikzcd}$$

Analogously, we can define:

$$CoP = \quad x \xleftarrow[f]{} z \xrightarrow[g]{} y$$

and define:

Definition 2.1.11. Let C be a category and $F : CoP \to C$ be a functor. Then the pushout of Fx and F_y denoted as $Fx \sqcup_{Fz} Fy$ is the colimit of the functor F.

Example 2.1.5.
- Fiber Product: The fiber product is the canonical pullback of the Top category. Let X, Y, Z be topological spaces and let $f : X \to Z$ and $g : Y \to Z$ be continous function. Then we define the fiber product as:

$$X \times_Z Y = \{(x, y) \in X \times Y : fx = gy\}.$$

  Note that, although it not usually noted, $X \times_Z Y$ depend on both f and g. In a diagram:

$$\begin{tikzcd} X \times_Z Y \arrow[r, "\pi_2"] \arrow[d, "\pi_1"'] & Y \arrow[d, "g"] \\ X \arrow[r, "f"] & Z \end{tikzcd}$$

  where π_i are the projection inherited from $X \times Y$. It is easy to check that, for any topological space Q and any $f' : Q \to X, g' : Q \to Y$ such that

$$\begin{tikzcd} Q \arrow[r, "f'"] \arrow[d, "g'"'] & Y \arrow[d, "g"] \\ X \arrow[r, "f"] & Z \end{tikzcd}$$

  there is an unique $u : Q \to X \times_Z Y$ that maps $q \mapsto (f'q, g'q)$ for every $q \in Q$.

- Seifert-Van-Kampen: This theorem is a classic result in Algebraic topology, that can be found in any classical source, such as [28]. It was independently discovered by Seifert [32] and Van-Kampen [36]. We present the result as in Brown's paper[6].

  We denote by $\pi_1(X, A)$ the fundamental groupoid of the topological space X over the set $A \subset X$. This notion was first showed by Brown. $\pi_1(X, X)$ would be the classical full fundamental groupoid, and $\pi_1(X, x_0)$ for some $x_0 \in X$ would be the fundamental group of X over x_0. Let us present the result.

  Theorem 2.1.3 (Seifert-Van-Kampen)**.** *Let the topological space X be covered by the interiors of two subspaces X_1, X_2 and let A be a set which meets each path component of X_1, X_2 and $X_0 = X_1 \cap X_2$, and $i_1 : X_0 \to X_i, j_i : X_i \to X$ be the canonical inclusions. Then A meets each path component of X and the following diagram is a pushout in the category of groupoids:*

$$\begin{tikzcd} \pi_1(X_0, A) \arrow[r, "\pi_1(i_1)"] \arrow[d, "\pi_1(i_2)"'] & \pi_1(X_1, A) \arrow[d, "\pi_1(j_1)"] \\ \pi_1(X_2, A) \arrow[r, "\pi_1(j_2)"] & \pi_1(X, A). \end{tikzcd}$$

Note the similarity of the pullback/pushout and the one of product/coproduct notation-wise. To understand these we have to consider the similarities of both construction. We are going to focus on the similarities of product and pullback, letting the coproduct/pushouts as duals.

In both case, the universal property consists of having an arrow to a generated object only if we have an arrow to each of its generators. In this line of reasoning, we can consider that the product is a pullback where we forget about the object z and its arrows. One easy way to generate that case is to consider the construction when Fz is a terminal object. In that case we have that the existence of Ff and Fg is a tautology, and we can consider

$$\begin{tikzcd}
 & q \arrow[ld, bend right, "q_1"'] \arrow[d, dashed, "u"] \arrow[rd, bend left, "q_2"] & \\
Fx & Fx \times_{Fz} Fy \arrow[l, "\pi_0"'] \arrow[r, "\pi_1"] & Fy
\end{tikzcd}$$

2.2 Adjoints

Adjointness is a fundamental theory within Category theory. It relates the homesets of two functors $F : C \to D, G : D \to C$. The importance of adjoints however, comes from it ubiquity among mathematics, often relate with the ubiquity of universality. This notion was first presented by Daniel M. Kan in [19]. For this section we follow the material presented in [24]. Adjoints are also called Adjunctions.

We will start by providing some intuitive notions, before a formal definition. Take the Forgetful Functor $U : Grp \to Set$, and the Free group Functor $\mathcal{F} : Set \to Grp$. We can see that it is not difficult to consider that we can compose F and G to make endofunctors. These endofunctors are far from being identities, nonetheless we have still a great relation between them: for every set X and group G, there is a function $f : X \to UG$ for each morphism $g : FX \to U$. Conversely, any function from X to UG induces a morphism.

Any avid reader will detect by this point the taste of universality. But there is a bit more, we have a bijective relation on every image home-set of both F and G. This is the underlying property of adjointness - to relate home-sets. Lets formalize this idea:

Definition 2.2.1. Let D, C be categories. And adjunction is a triple $(F, G, \varphi) : D \to C$ where F, G are functors:

$$\begin{tikzcd}
D \arrow[r, shift left, "G"] & C \arrow[l, shift left, "F"]
\end{tikzcd}$$

while φ is a transformation that maps each $(c, d) \in Ob(C \times D)$ with a bijection:

$$\varphi_{c,d} : \hom_A(Fd, c) \equiv \hom_X(d, Gc).$$

which is natural in c and d.

Remark 2.2.1. but we will stick with the notation of F being the *left adjoint* and G being the *right adjoint*, denoted as $F \dashv_\varphi G$ or just $F \dashv G$. When a functor *is* a left (resp. right) adjoint it *has* a right (resp. left) adjoint.

We have a bit to unpack in this definition. We will start to understand that $\varphi : Hom_C(F\cdot,\cdot) \to \hom_D(\cdot, G\cdot)$ is a natural bijection in each variable. That is, for every $f : d \to d' \in C, k : c \to c' \in D$ the following diagrams commutes.

$$\begin{tikzcd} \hom_C(Fd,c) \arrow[r, "\varphi_{c,d}"] \arrow[d, "\hom_C(Fd,\cdot)(g)"] & \hom_D(d,Gc) \arrow[d, "\hom_D(d,G\cdot)(g)"] \\ \hom_C(Fd,c') \arrow[r, "\varphi_{c',d}"] & \hom_D(d,Gc') \end{tikzcd} \qquad \begin{tikzcd} \hom_C(Fd,c) \arrow[r, "\varphi_{c,d}"] \arrow[d, "\hom_C(F\cdot,c)(f)"] & \hom_D(d,Gc) \arrow[d, "\hom_D(\cdot,Gc)(f)"] \\ \hom_C(Fd,c') \arrow[r, "\varphi_{c,d'}"] & \hom_D(d,Gc') \end{tikzcd}$$

The definition of adjoint can be reworked as:

Proposition 2.2.1. An adjoint can be described as a bijection that send each $f : Fx \to a \in A$ to $\varphi(f) : x \to Ga$ such that, for every $h : x \to x' \in X, k : a \to a' \in A$:

$$\varphi(k \circ f) = Gk \circ \varphi(f), \qquad \varphi(f \circ Fh) = \varphi(f) \circ h.$$

Proof. These conditions ensures the naturality of the bijection. □

Remark 2.2.2. It is equivalent to require φ^{-1} to be natural.

This proposition, that can be obviated and is rarely used by itself, is really important as it encapsulate the idea behind the commonplace concepts of *unit* and *counit*. The key reasoning behind these objects is to consider what would happen whenever f is the identity, following the notation of the previous proposition. This provide us of full control of the bijection based only on arrow $\varphi(f)$. This is also the fundamental idea behind proposition 2.1.1.

We have already suggested the relationship between universality and adjointness. We can summary that relation in the following property:

Proposition 2.2.2. An adjunction $(F, G, \varphi) : D \to C$ determines:

- a natural transformation $\eta : I_D \to GF$ such that $\eta(d) : d \to GFd$ is universal to G from d for every $d \in Ob(D)$. Conversely, we can define:

$$\varphi(f : Fd \to c) = Gf \circ \eta(d) : d \to Gc.$$

- a natural transformation $\epsilon : FG \to I_C$ such that $\varepsilon(c) : FGc \to c$ is universal from c to F and we can define:

$$\varphi(g : d \to Gc) = \varepsilon(c) \circ Fg : Fd \to c.$$

- We have that the natural transformation $\eta_{G\cdot} \circ G(\varepsilon_\cdot)$ is the identity natural transformation.

Remark 2.2.3. We can therefore consider an adjoint (F, G, φ) as a quintuple $(F, G, \varphi, \eta, \varepsilon)$. η is usually called the *unit* and ε is called the *counit*.

Proof.
- Let $a = Fd$, then we have a natural bijection φ:

$$\hom_C(a, c) \equiv \hom_D(d, Gc).$$

By proposition 2.1.1 we have an universal arrow $\eta_d : d \to Ga = GFd$ is provided, where $\eta_d = \varphi(1_a)$. The function $d \to \eta(d)$ is natural $I_D \to GF$ as the following diagram commutes due to the naturality of φ:

$$\begin{tikzcd} d \arrow[r, "\eta c"] \arrow[d, "h"] & GFd \arrow[d, "GFh"] \\ d' \arrow[r, "\eta c'"] & GFd''. \end{tikzcd}$$

- Analogous.
- $1_G a = G(\varepsilon_a) \circ \eta_{Ga}$.

□

From this proposition we can see that we may be able to define an adjoint based only on the universal arrows provided. We can summary a few equivalent definition of adjoint:

Proposition 2.2.3. Each adjoint $(F, G, \varphi, \eta, \varepsilon) : D \to C$ is completely determined by the items in any one of the following list:

1. F, G and a natural transformation $\eta : 1_D \to GF$ such that $\eta(d)$ is universal for every $d \in Ob(D)$.
2. F, G and a natural transformation $\varepsilon : FG \to 1_C$ such that $\varepsilon(c)$ is universal for every $c \in (C)$.
3. G and for each $d \in Ob(D)$, an object $F_0(d) \in Ob(C)$ and an universal arrow $\nu(x) : d \to GF_0 d$.
4. F and for each $c \in Ob(C)$, an object $G_0(c) \in Ob(D)$ and an universal arrow $\varepsilon(c) : c \to GF_0 c$.
5. F, G and two natural transformation $\eta : I_D \to GF$, $\varepsilon : FG \to I_C$ such that both composites are identities.

Proof. 1. We need to define the natural bijection $\varphi : \hom_C(F\cdot, \cdot) \to \hom_D(\cdot, G\cdot)$. Let for each $f : Fd \to c$:

$$\varphi(g) = Gf \circ \eta(c) : d \to Gc$$

Is well defined and it is a bijection due to $\eta(d)$ universality.

It is natural in d because η is natural and is natural in c because G s a functor.

2. Dual of the previous.
3. We will proof that there is only one functor with F_0 as object function such that $\eta(d) : I_D \to GF$ is natural. The naturality and universality told us that each $h : d \to d' \in Ob(D)$ induces two arrows:

$$\begin{tikzcd} F_0 d \arrow[d, dashed] & d \arrow[r, "\eta(d)"] \arrow[d, "h"] & GF_0 d \arrow[d, dashed] \\ F_0 d' & d' \arrow[r, "\eta(d')"] & GF_0 d' \end{tikzcd}$$

So the only way to define $Fh = \eta(d) \circ h$. We conclude applying 1.

4. Dual of the previous.

5. We use η and to define two functions:

$$\hom_C(Fd, c) \underset{\sigma}{\overset{\tau}{\rightleftarrows}} \hom_D(d, Gc)$$

such that $\tau(f) = Gf \circ \eta_d$ and $\sigma(g) =_d \circ Fg$. as both composites are the identity, $\tau \circ \sigma$ and $\sigma \circ \tau$ are the identities. It is clearly natural due to proposition 2.2.1.

□

To further illustrate this concept we present some examples:

Example 2.2.1.
- Free category from a graph and forgetful functor: Given a graph G, we can define the free category $\mathcal{F}(G)$ by adding the arrows needed for the graph to satisfy the category axioms, that is:
  - $\mathcal{F}(G)$ has as objects all nodes in G.
  - $\mathcal{F}(G)$ has as arrows all identities in G, and every finite composition $f = f_1 \circ f_2 \circ ... \circ f_n$ where f_i is either an identity or an arrow in G. Note that the concept of arrow composition in graph can be applied directly from it categorical definition.

  Then, we can define a functor $\mathcal{F} : Graph \to Cat$ that maps every (small) graph G to $\mathcal{F}(G)$ and every graph morphism φ to a category morphism, that take

  $$f = f_1 \circ f_2 \circ ... \circ f_n \mapsto (\mathcal{F}\varphi)f = \varphi(f_1) \circ \varphi(f_2) \circ ... \circ \varphi(f_n).$$

  We can see that we have an adjoint $F \dashv U$ where the unit is given by the insertion of generators.
- Čech compactification[3]: We have presented in 1.2.2 the $\beta : Hauss \to Comp$ Stone-Čech functor. If we consider the injection of categories $i : Comp \to Hauss$ we can check that is an adjoint $\beta \dashv i$.

2.2.1 Equivalence of Categories

We can define an isomorphism of categories the same way that we define isomorphisms in any other category: an isomorphism is an arrow (Functor) with a two sided inverse. This definition, while standard, is quite restrictive. We are going to need some more lax concept of equivalence of categories, in the sense that while not being exactly the same, they are mostly the same. Formally:

Definition 2.2.2. A functor $F : C \to D$ is an equivalence of categories if there exists a functor $G : D \to C$ and two natural isomorphisms $\varphi : I_C \to GF$ and $\zeta : I_D \to FG$

For the canonical example we may introduce a really organic concept: the *skeleton* of a category. Quite often in mathematics we do not consider all objects of certain type, but rather objects up to isomorphism. The skeleton category of a category C is another category where we consider the objects up to isomorphism. Formally:

Definition 2.2.3. Let C be a category. The skeleton of C, namely $ske(C)$, is a subcategory such that every object in C is isomorphic to an object in $ske(C)$.

The definition of equivalence of categories previously outline will allow us, for example, to consider $C \sim ske(C)$, while not being isomorphic. Continuing our discussion, it is not difficult to draw similarities between equivalences and adjoints. That motivates the further definition:

Definition 2.2.4. Let $(F, G, \varphi, \eta, \varepsilon)$ be an adjoint. It is called an *adjoint equivalence* whenever η and ε are natural equivalences.

It is clear that every adjoint equivalence induces two equivalences in F and G. We state this idea in the following proposition:

Proposition 2.2.4. [24, Theorem 1, 4.4] Let $F : C \to D$ be a functor. Then the following properties are equivalent:

i) F is an equivalence of categories.

ii) F is part of an adjoint equivalence.

iii) F is full and faithfull, and each object $d \in D$ is isomorphic to Fc for some object $c \in C$.

Proof.

$ii) \implies i)$ Given an equivalent adjoint $(F, G, \varphi, \eta, \varepsilon)$, then F is an equivalence with G, η, ε.

$i) \implies iii)$ $\varphi : GF \equiv I_C$ implies that for every $c \in Ob(C)$ we can consider $c \equiv GFc$. Then considering the natural isomorphism $\zeta : FG \equiv I_D$ states for every $f : a \to a' \in D$

$$\begin{tikzcd}
FGa \arrow[d, "FGf"] & a \arrow[l, "\zeta^{-1}a"'] \arrow[d, "f"] \\
FGa' \arrow[r, "\zeta a'"] & a'
\end{tikzcd}$$

there is one FGf for each f and we get that G is faithful. Faithfullnes of F can be prove simetrically. To see that G is full we can consider any $h : Ga \to Ga'$ and defining $f = \zeta a \circ h \circ \zeta a'$ we follow hat $FGf = Gh$. As T is faithfull, $Gf = h$ so that G is full. Procees simetrically with F.

$iii) \implies ii)$ We need to construct G so that F is a left adjoint. The idea is to define η and apply the previously developed characterizations.

Due to the fully faithfulness of F we can choose for every $d \in D$ an object $c \in C$ such that there is an isomorphism $\eta d : d \to Fc$ and for each $c' \in C, f : d \to Fc' \in D$ there is some $g \in Ar(C)$ (that exists because F is full and is unique because of faithful) such that:

$$\begin{tikzcd}
d \arrow[r, leftrightarrow, "\eta d"] \arrow[dr, "f"'] & Fc \arrow[d, "Fg"] \\
& Fc'.
\end{tikzcd}$$

We define $G_0 d = c$. Also note that ηd is universal from d to F. Therefore, F is part of an adjoint by proposition 2.2.3 F is part of an adjoint $(F, G, \eta, \varepsilon)$. As with every adjoint, $F(\varepsilon c) \circ \eta(Fc) = 1_{Gc}$. Thus $F(\varepsilon c)$ is invertible and by fully faithfulness of F so is ϵc, thus having an adjoint equivalence.

$\square$

In the next few subsection, we are going to introduce categories, with some additional structures onto them. This type of considerations, are commonplace in category theory, and will provide useful considerations.

2.2.2 Closed Cartesian Categories

The notion of product seen in 2.1.8 is a notion that seek to catch the essence of what the Cartesian product is in the category of Set. In this category it happens that for any two objects, one can consider the product without any hesitance about it existence. A *Cartesian category* will be a category that, in some sense, maintain this property (of being closed under Cartesian product). Formally:

Definition 2.2.5. A Cartesian category is a category with a specified terminal object T and for which every finite product exists.

Remark 2.2.4. Finite product are equivalent to binary product.

Nonetheless, further on the text we will need categories with some even nicer properties. These categories are called *closed catersian categories*. We will define them formally first, and after unpack the information that the definition contains:

Definition 2.2.6. A C is closed cartesian category, ccc for short, if each of the following functors:

$$\begin{aligned} F_1 : C &\to 1 \\ c &\to e \end{aligned} \qquad \begin{aligned} F_2 = \Delta : C &\to C \times C \\ c &\to (c, c) \end{aligned} \qquad \begin{aligned} F_3^b : C &\to C \\ c &\to c \times b \end{aligned}$$

has a *specified* right adjoint, denoted by:

$$\begin{aligned} G_1 : 1 &\to C \\ e &\to t \end{aligned} \qquad \begin{aligned} G_2 : C \times C &\to C \\ (c, b) &\to c \times b \end{aligned} \qquad \begin{aligned} G_3^b : C &\to C \\ c &\to c^b \end{aligned}$$

where there is one $F_3^b \dashv G_3^b$ for every $b \in (C)$.

These adjoints provide us with a lot of information:

- From F_1 being an adjoint we get that t is terminal as for every $c \in Ob(C)$:

$$\{id_e\} \equiv \hom_1(F_1(c), e) \equiv \hom_C(c, Ge = t)$$

- From F_2 we get that every pair c, b has a product at is states the universal property of product:

$$\hom_{C\times C}(F_2(c), (c', b')) \equiv \hom_{C\times C}((c, c), (c', b')) \equiv \hom_C(c, c' \times b')$$

  that is, for every object in $c \in C$, to provide a morphism to $c \to c' \times b'$ is the same that providing to morphisms $c \to c', c \to b'$.

- First of all, as every product exists, and it specified by F_2, we can define F_3^b for every b. Then, it adjointness states that:

$$\hom_C(c \times b, d) \equiv \hom_C(c, d^b)$$

  This object d^b is called the exponential object or map object. Note that to fully define the adjoint we can take one of two steps, the functor G_3^b is defined only on the objects:

  - We can define how G_3^b acts on arrows.
  - We can provide an arrow ε:

$$\varepsilon : F_3^b G_3^b(c) = c^b \times b \to c = I(c)$$

    that is natural in c and universal from F_3^b to c.

The intuitive notion is that this object represent in some way the arrows from b to c, being a generalization of the function set of two small sets. The best way to understand this adjoint is by the natural transformation $\varepsilon^b(c^b \times b) = c$. In this context, this is called *evaluation arrow*. In the context of *Set* this amount to the classic evaluation of a function.

Remark 2.2.5. Note that a Closed Cartesian category grow in the notion of a cartesian category. It is in fact a Cartesian category on which we have a function object, with an evaluation arrow that is universal.

Cartesian Closed Categories will be reformulated in Chapter 4, proposition 4.1.2. We will use this last ideas in other to equationally present the concept. For now on, lets provide some examples of Cartesian Closed Categories:

Example 2.2.2.
- *Set*. We have the set $\{*\}$ as a terminal object, products as in example 2.1.2. Given two sets C, B we can define the exponential object $C^B = \hom(B, C)$. The adjointness is then implied by the currification process:

$$g : A \times B \to C \mapsto f : A \to B^C,$$

where $f(a)(\cdot) = g(a, \cdot)$.

- We will prove in chapter 4 that λ-calculus is, in some sense, a closed cartesian category.

We can consider the duals:

Definition 2.2.7. A *cocartesian* category is a category with a specified initial object $\perp$ and for which every finite coproduct exists. A *bicartesian* category is a category that is both cartesian and cocartesian.

Definition 2.2.8. A *closed bicartesian category* is a bicartesian category that is also a closed cartesian category.

As with almost every structure, a category can be deduced is the one of the objects along with structure-preserving functors.

Definition 2.2.9 (Category of closed cartesian categories)**.** We can define the *category of closed cartesian categories Cart* that has as objects all small ccc, and as objects all functors that preserve the specified terminal, products and exponential objects.

The morphism of cccs are often called *cartesian closed functors.*

Part II

Lambda Calculus

Chapter 3

Lambda Calculus

In this chapter we define the λ-calculus as a formal language of computation, and on this define the concept of typing, ubiquitous in programming. The main references for the general approach if this chapter are are [33] and [17]. For the historical motivation for the development of the theory we refer to [7].

In the 19th century mathematicians developed the concept of function. In this time it was question whether two functions were the same. The most widespread view is the *extensional* approach. This approach consider two functions to be the same whenever for the same input they have the same output. A function f as a pairing of an X domain to an Y codomain. These functions can be considered as sets $f \subset X \times Y$.

Despite that, this notion can be considered misleading. For example, let p be a prime big enough, and let f and g be two endomorphisms of $\mathbb{Z}/p\mathbb{Z}$. One could argue that the endomorphism of $f(x) \to x^2$ is different to $g(x) \to \log_a a^{x+2}$. Despite having the same output for the same input, they are clearly different in *complexity*, with complexity understood as the cost of computing a function. It even involve the resolution of a discrete logarithm, that is highly non trivial.

This view, in which not only is the result of the function important but how is that result obtained, is called the *intensional* approach. This approach, gain traction in early 1930 as Church[9], Gödel[1] or Turing[35] start formalizing what is to be computable. Nowadays, after the wake of computation, the intensional approach have come to be as relevant as the extenisional approach.

3.1 Untyped λ-Calculus

We will start defining the most simple version of λ-calculus: untyped λ-calculus. Let's define some concepts:

1. An alphabet A is an arbitrary, maybe finite, non-empty set.
2. A symbol a in an element of the alphabet.
3. A word is a finite sequence of symbols.
4. The collection of all possible finite words over an alphabet A is denoted by A^*.
5. A language L over A is a subset of A^*.

There are a lot of languages. For example, Spanish is a language, with a well-known alphabet L, with a proper subset of words over L^*.In the same fashion, we define λ-calculus as a formal language, defining its syntax, that is, what words are valid.

Syntax of untyped λ-calculus

We start with the basic building blocks, which collectively form what is called the alphabet:

- We use $x, y, z, ...$ to denote variables. As more variables are necessary sub-indexes will be used, up to countable infinite variables.
- We consider an abstract connector λ.
- Auxiliary, we consider the characters ".", "(" and ")".

Now, we are ready to formally define the untyped λ-calculus:

Definition 3.1.1 (Syntax of Untyped λ-calculus (section 2.1 [33])). A λ-calculus term (sometimes called formula) is defined inductively:

- Every variable $x, y, z...$ is a valid formula.
- If A, B is a formula, then AB is a valid formula.
- If A is a formula and x is a variable, then $\lambda x.M$ is a valid formula.

The set of all variables is denoted by $\mathcal{V}$ and the set of all λ-formulas is denoted by Λ.

Remark 3.1.1. Λ is countable.

Dealing with formal languages we make use of similar inductive statements more often than not. we find it useful to introduce the Backus-Naur Form notation [20], BNF for short. A BNF specification is a set of derivation rules, written as

$$\text{word1}, \text{word2} \ldots ::= \text{expression1} \mid \text{expression2} \mid \ldots,$$

where each word is a generic valid word from the language, and each expression consists of a derived valid formulas. Expressions are separated by the vertical bar: $|$. For example, we can revisit definition 3.1.1 as follow:

Definition 3.1.2. The formulas of λ-calculus are built via the BNF:

$$A, B ::= x \mid (AB) \mid (\lambda x.A).$$

where x denote any variable in $\mathcal{V}$.

Remark 3.1.2. Note that there is no need to specify the alphabet aside from the set $\mathcal{V}$.

From this point on we know how the λ-terms are constructed. This point is better understood with some examples, using natural numbers. We recommend to trust us in their existence, and use a naive intuition of what a natural is when reading this examples. The notion of natural numbers is formalized definition 3.1.17 and revisited in section 4.2.1.

3.1.1 Reductions in untyped λ-calculus

Let us now explain the idea behind the formalism. Consider the expression $\lambda x.(x+1)$. This expression represent the idea of the function $f(x) = x + 1$ that take a variable x and return $x + 1$. $\lambda x.M$ is called the *abstraction* of x.

From the notion of abstraction naturally arises the second one: *application*. Consider terms $M = \lambda x.x + 1$ and $N = 3$. Then $MN = (\lambda x.x + 1)(3)$ represent the application of M to N. In untyped λ-calculus, N can be any term. Thus for example, the term $\lambda f.\lambda g.fg$ just represent the composition of terms.

Example 3.1.1. The term

$$\lambda g.(\lambda f.(\lambda x.(gf)(x)))(x+1)(x+2),$$

can be understood as $(g \circ f)(x)$ where $g(x) = x+1$ and $f(x) = x+2$.

In a expression $M = (\lambda x.N)$ we say that the variable x is *bound* in M.

The idea of α-equivalence, $=_\alpha$, is that expressions such as $\lambda x.x$ and $\lambda y.y$ are essentially the same. That is, we consider terms up to *rename* of variables. To formalise this, we have to formalize the concept of free and bound variables, and the concept of renamed.

Definition 3.1.3. We have a *free variable* function $FV : \Lambda \to \mathcal{P}(\mathcal{V})$ defined recursively:

- $FV(x) = \{x\}$ for every $x \in \mathcal{V}$.
- $FV(MN) = FV(M) \cap FV(N)$ for every $M, N \in \Lambda$.
- $FV(\lambda x.M) = FV(M) \backslash \{x\}$ for every $M \in \Lambda, x \in \mathcal{V}$.

Given a term M, if $x \in FV(M)$ we say that x is a free variable in M or that M has a free variable x.

Definition 3.1.4. We say that a term is *closed* if it has no free variables.

We can now define process of *substitution*, and define a rename as a particular case. This process is the one behind the intuitive idea of evaluation. For example when we consider $(\lambda y.y^2 + y)(4) = 4^2 + 4 = 20$ we are replacing the value of x by the term 4.

Definition 3.1.5. The substitution of N for free occurrences of x in M, denoted as $M[N/x]$ is defined recurrently in the structure of λ-terms by:

$$\begin{aligned}
x[N/x] &\equiv N, && \\
y[N/x] &\equiv y, && \text{if } x \neq y, \\
(MP)[N/x] &\equiv (M[N/x])(P[N/x]), && \\
(\lambda x.M)[N/x] &\equiv \lambda x.M, && \\
(\lambda y.M)[N/x] &\equiv \lambda y.(M[N/x]), && \text{if } x \neq y \text{ and } \notin FV(N), \\
(\lambda y.M)[N/x] &\equiv \lambda y'.((M[y'/y])[N/x]), && \text{if } x \neq y \text{ and } y \in FV(N), \\
& && \text{with } y' \notin FV(N) \cup \{x\}.
\end{aligned}$$

When $N = y$ a variable we say that $[N/x] = [y/x]$ is a rename.

Definition 3.1.6. We define the α-equivalence $=_\alpha$ as the smallest congruence relation on Λ such that:

$$\lambda x.M =_\alpha \lambda y.M[x/y], \qquad \forall y \in \mathcal{V} \backslash FV(M).$$

In other words is the congruence that consider every rename of a bounded variable as equals. This type of property-oriented definition is commonplace in λ-calculus, as it allows for some synthetic and goal-oriented definition. More often than note we will not give explicit use of this equivalence.

Using this same tool we are going to define the β-reduction. This is not going to be an equivalence implication, but rather a relationship. It abstracts the notion of 4 and $(\lambda x.2 + x)(2)$ being equals. Formally:

Definition 3.1.7 (Section 2.5 [33]). We define the *single-step β-reduction* as the smallest relationship $\to_\beta$ such that:

$$
\begin{array}{ll}
(\beta) & \dfrac{}{(\lambda x.M)N \to_\beta M[N/x]}, \\
(\text{cong}_1) & \dfrac{M \to_\beta M'}{MN \to_\beta M'N}, \\
(\text{cong}_2) & \dfrac{N \to_\beta N'}{MN \to_\beta MN'}, \\
(\zeta) & \dfrac{M \to_\beta M'}{(\lambda x.M) \to_\beta \lambda x.M'}.
\end{array}
$$

Remark 3.1.3. In this definition we can see that rule (β) is the main objective, while the others are necessary so that it maintain the structure.

Definition 3.1.8. We define the *multiple-step β-reduction* $\twoheadrightarrow_\beta$ as the reflexive, transitive closure of $\to_\beta$.

Definition 3.1.9. We define the β-equivalence $=_\beta$ as the symmetric closure of $\twoheadrightarrow_\beta$.

Up to this point the focus of the system was to define an intensional language for computation. β-reduction encapsulate the concept of computation, where we have evaluation.

Example 3.1.2.
- Ending example.
- Never ending example.

Should we want to consider a way of having an extensional approach, for example, to generate normal forms for the terms, we would need more machinery. The η-equivalence provides us with the tools to consider λ-calculus in a extensional way.

Definition 3.1.10. We define the single-step η-reduction $\to_\eta$ as the smallest relationship such that:

$$
\begin{array}{lll}
(\eta) & \dfrac{}{(\lambda x.Mx) \to_\eta M} & \forall x \notin FV(M), \\
(\text{cong}_1) & \dfrac{M \to_\eta M'}{MN \to_\eta M'N}, & \\
(\text{cong}_2) & \dfrac{N \to_\eta N'}{MN \to_\eta MN'}, & \\
(\zeta) & \dfrac{M \to_\eta M'}{(\lambda x.M) \to_\eta \lambda x.M'}. &
\end{array}
$$

Similarly, we define the multiple-step η-reduction $\twoheadrightarrow_\eta$ as the transitive reflexive closure of $\to_\eta$, and the η-equivalence as the symmetric closure of $\twoheadrightarrow_\eta$.

Definition 3.1.11. We define the single-step $\beta\eta$-reduction $\to_{\beta\eta}$ as the union of $\to_\beta$ and $\to_\eta$. We define the multiple-step $\beta\eta$-reduction $\twoheadrightarrow_{\beta\eta}$ as the transitive reflexive closure of $\to_{\beta\eta}$.

Name	Main rule	Equivalence
α	$(\lambda x.M) =_\alpha \lambda y.M[x/y]$	Yes
β	$(\lambda x.M)N \to_\beta M[N/x]$	No
η	$(\lambda x.Mx) \to_\eta M$	No

TABLE 3.1: Reductions

3.1.2 Church-Rosser Theorem

This subsection we present an important result for λ-calculus: the *Church-Rosser* theorem. The idea behind this theorem is to prove that every reduction (either β, η or a mix) provide an unified sense of reduction. First, we present some definitions.

Definition 3.1.12. Consider a relation $\to$ and let $\twoheadrightarrow$ be its reflexive transitive closure. We can define three relations:

1. The Church-Rosser Property:

$$M \twoheadrightarrow N, M \twoheadrightarrow P \implies \exists Z : N \twoheadrightarrow Z, P \twoheadrightarrow P.$$

2. The Quasidiamond Property:

$$M \to N, M \to P \implies \exists Z : N \twoheadrightarrow Z, P \twoheadrightarrow P.$$

3. The Diamond Property :

$$M \to N, M \to P \implies \exists Z : N \to Z, P \to P.$$

Remark 3.1.4. Note that, while 3) $\implies$ 1), it is not necessary that 2) $\implies$ 1).

With this notation, we introduce the Church-Rosser Theorem, proved as done by Martin-Löf.

Theorem 3.1.1. *Church-Rosser*

1. $\twoheadrightarrow_{\beta\eta}$ *satisfy the Church-Rosser property.*
2. $\twoheadrightarrow_\beta$ *satisfy the Church-Rosser property.*

We prove the theorem only for the $\twoheadrightarrow_{\beta\eta}$ case. We refer to [Theorem 1.32, [17]] for a full proof for $\twoheadrightarrow_\beta$. he first step in the proof is going to be an alternative definition of $_{\beta\eta}$, as a completion of a new reduction $\triangleright$.

Definition 3.1.13 (parallel one-step reduction)**.** We define the parallel one-step reduction $\triangleright$ as the smallest relationship such that,

$$(1) \qquad \frac{}{x \triangleright x}$$

$$(2) \qquad \frac{M \triangleright M' \qquad N \triangleright N'}{MN \triangleright M'N'}$$

$$(3) \qquad \frac{M \triangleright M'}{(\lambda x.M) \triangleright \lambda x.M'}$$

$$(4) \qquad \frac{M \triangleright M' \qquad N \triangleright N'}{(\lambda x.M)N \triangleright M'[N'/x]}$$

$$(5) \qquad \frac{M \triangleright M'}{(\lambda x.Mx) \triangleright M'} \qquad \forall x \notin FV(M)$$

where x is any variable and M, N, M', N' any term.

Lemma 3.1.2 (Characterization of $\triangleright$)**.** *Let M, M' be terms, then:*

1. $M \rightarrow_{\beta\eta} M' \implies M \triangleright M'$
2. $M \triangleright M' \implies M \twoheadrightarrow_{\beta\eta} M'$

Proof. 1. We apply induction on the structure of $\rightarrow_{\beta\eta}$. That is, we know that $\rightarrow_{\beta\eta}$ have to be constructed following a series of rules. We are going to use induction on the last rule used. Therefore we conclude every rule use for $\rightarrow_{\beta\eta} M'$ implies $M \triangleright M'$.

(β) In this case, then $M = (\lambda x.Q)M$ and $M' = Q[N/x]$. Then using (4) $M \triangleright M'$.

(η) In this case, then $M = (\lambda x.Qx)$ and $M' = Q$. Then using (5) $M \triangleright M'$.

(cong_1) In this case, then $M = PQ$ and $M' = PQ'$ for some $Q \rightarrow_{\beta\eta} Q'$. Using induction $Q \triangleright Q'$ and using (5) $M \triangleright M'$.

(cong_2) Analogous.

(ζ) In this case, then $M = \lambda x.Q$ and $M' = \lambda x.Q'$ for some $Q \rightarrow_{\beta\eta} Q'$. Using induction $Q \triangleright Q'$ and using (3) $M \triangleright M'$.

Where, in every point, x denote some variable, and M, N, M', N', P, Q, Q' denote some term.

2. Similarly, every possible in which $M \triangleright M'$ rule derive in $M \twoheadrightarrow_{\beta\eta} M'$:

(1) By reflexivity of $\twoheadrightarrow_{\beta\eta}$.

(2) By $\text{cong}_1, \text{cong}_2$ in either definition of $\rightarrow_\beta$ and $\rightarrow_\eta$.

(3) By ζ in either definition of $\rightarrow_\beta$ and $\rightarrow_\eta$.

(4) Then we have $(\lambda x.M)N \triangleright M'[N'/x]$ with $M \triangleright M'$ $N \triangleright N'$. By induction $M \twoheadrightarrow_{\beta\eta} M'$ and $N \twoheadrightarrow_{\beta\eta} N'$. By transitive closure:

$$(\lambda x.M)N \twoheadrightarrow_{\beta\eta} (\lambda x.M')N \twoheadrightarrow_{\beta\eta} (\lambda x.M')N' \twoheadrightarrow_{\beta\eta} (M')[N'/x].$$

(5) Then we have $(\lambda x.Mx) \triangleright M'$, for some $M \triangleright M'$ and for some $x \notin FV(M)$. Finish using (cong_2) and η.

Where, in every point, x denote some variable, and M, N, M', N', P, Q, Q' denote some term. $\square$

Lemma 3.1.3 (Subtitution Lemma)**.** *If $M \triangleright M'$ and $U \triangleright U'$, then $M[U/y] \triangleright M'[U'/y]$.*

Proof. As we define in 3.1.5 a capture-avoiding substitution, in the sense that we do not let bound variable to be substituted, we can assume without any loss of generality that M. Similarly to previous lemma, we proceed by induction, studying the last rule applied. The induction is proceed on the rules applied to $M \triangleright M'$.

(1) In this case $M = M' = x$. If $x \neq y$ the substitution does not alter M so we have finished. If $x = y$, the term is only $U \triangleright U'$, that we know by hypothesis.

(2) In this case $M = NP, M' = P'N'$ for some terms N, N', P, P' such that $N \triangleright N'$, $P \triangleright P'$. Proceed by induction on these implications and apply (2).

(3) In this case $M = \lambda x.N$ and $M = \lambda x.N'$, for some $N \triangleright N'$. Apply induction on N and end by (3).

(4) Then we have $(\lambda x.M)N \triangleright M'[N'/x]$ with $M \triangleright M'$ and $N \triangleright N'$. By induction on both of the relationship, and applying (4).

(5) Then we have $(\lambda x.Mx) \triangleright M'$, for some $M \triangleright M'$ and for some $x \notin FV(M)$. Finish using induction on $M \triangleright M'$ and by (5).

□

While the proof is rather straight forward, one can see that it does require induction, and thus the length of it. [1]

Definition 3.1.14 (Maximal parallel one-step reduction)**.** Given a term M, the *maximal parallel one-step reduction* M^* is defined inductively:

- $(x)^* = x$
- $(PN)* = P^*N^*$ but for β-reductions.
- $((\lambda x.P)N)^* = Q^*[N/x]$
- $(\lambda x.N)^* = \lambda x.N^*$ but for η-reductions,
- $(\lambda x.Nx)^* = N^*$

where x is any variable, and N, P is any term.

Lemma 3.1.4 (Maximal Parallel one-step)**.** *If $M \triangleright M'$ then $M' \triangleright M^*$.*

Proof. We proceed by induction on M again:

(1) In this case $M = M' = x$. Then $M^* = x$.

(2) In this case $M = NP, M' = P'N'$ for some terms N, N', P, P' such that $N \triangleright N'$, $P \triangleright P'$.

- If NP is not a β-reduction, then $M^* = P^*N^*$ and we can use the hypothesis induction on N^* and P^* and (2).
- If NP is a β-reduction, then $N = \lambda x.Q$, thus $M^* = Q^*[N^*/x]$. $P = \lambda x.Q$ and $P \triangleright P'$ could be derived using congruence to λ-abstraction (2) or extensionality (5). In the first case use induction on Q and (4). On the second case $Q = Rx$, use induction on R, and apply substitution lemma.

Proceed by induction on these implications and apply (2).

(3) In this case $M = \lambda x.N$ and $M = \lambda x.N'$, for some $N \triangleright N'$. If M is not an η-reduction, then use induction hypothesis and finish with (3). Otherwise we have that $N = Py$, and distinguish two cases on the last rule applied to N:

(2)$\to$ (3) Then apply induction hypothesis to P and end using (5).

(4)$\to$ (3) We have that $N = P = \lambda y.Q \triangleright N' = Q'[x/y]$. Apply induction hypothesis using that $M' = \lambda x.N' = \lambda x.Q'[x/y] = \lambda y.Q'$.

(4) Then we have $(\lambda x.P)N \triangleright P'[N'/x]$ with $P \triangleright P'$ and $P \triangleright P'$. By substitution lemma.

(5) Then we have $(\lambda x.Px) \triangleright P'$, for some $P \triangleright P'$ and for some $x \notin FV(P)$. Finish using induction on

□

Proof of Theorem 3.1.1. If a relation $\to$ satisfy the diamond property, its reflexive, transitive closure $\twoheadrightarrow$ satisfy the Church-Rosser property. As a consequence of lemma 3.1.4, we know that $\triangleright$ satisfy the diamond property. By lemma 3.1.2 we know that the reflexive, transitive closure of $\triangleright$ is $\twoheadrightarrow_{\beta\eta}$. □

[1] Maybe add some explanation here.

Now let take a moment to comment on the usefulness of each reduction. We can say that β reduction is the soul of computation while η is useful to cleanup the results. This can be used to define *normal* forms for untyped lambda-terms. More infomation on normalization can be found on [33, Section 7] and [4].

3.1.3 Fixed points and Programming

In this subsection we expose the basic rudiments for programming in untyped λ-calculus. In particular, we explain how to define, for example, a booleans, an integer type and a recursion operator, thus showing the computational potential of recursively enumerable languages. We start with the booleans.

Definition 3.1.15 (Boolean in untyped lambda-calculus)**.** We define the two values T and F, true and false, as:

$$\begin{aligned} \text{True} &= \lambda xy.z, \\ \text{False} &= \lambda xy.z. \end{aligned}$$

We also define the operations:

$$\begin{aligned} \text{Not} &= \lambda a.a\,\text{False}\,\text{True}, \\ \text{And} &= \lambda ab.ab\,\text{False}, \\ \text{Or} &= \lambda ab.a\,\text{True}\,b. \end{aligned}$$

We can check that this construction left us with the basis of a boolean logic, after checking the truth values of the different operations provided. This definition of truth value is really convenient, as it allow us to easily implement control flow of programs.

Definition 3.1.16. We define the $\text{If} = \lambda x.x$.

We can see that in this case, $\text{If}\,\text{True}\,MN = M$ and $\text{If}\,\text{False}\,MN = N$. This construction is very natural and widely used in computer programs. Next, we will define Naturals number.

Definition 3.1.17 (Natural numbers in untyped lambda-calculus)**.** Let f, x be fixed λ-terms, and write $f^n x = f(f(f(...(f(x))...)))$. Then, for each $n \in \mathbb{N}$, we define the *nth Church numeral* as $\overline{n} = \lambda f x. f^n x$.

Finally, we consider the notion of recursive function. This is done with a elegant artefact, based on an idea of fixed points. Let's begin:

Theorem 3.1.5 (Fixed points)**.** *For every term F in untyped λ-calculus, there is a fixed point.*

Proof. Let $A = xy.y(xxy)$ and $\Theta = AA$. Then, for every lambda-term F we have that $N = \theta F = FN$, thus being a fixed point. In fact:

$$N = \Theta F = AAF = (\lambda xy.y(xxy))AF \twoheadrightarrow_\beta F(AAF) = F(\Theta F) = FN.$$

$\square$

The proof of this theorem led to a new definition:

Definition 3.1.18. The term Θ used in the previous theorem is called *Turing fixed point combinator*.

This existence of fixed point theorem is really useful, as we can now define *recursion*. The idea is to define a function as a term with itself as a parameters and proceed to evaluate a fixed point. Let us first present an example.

Definition 3.1.19. We define the terms:

$$\begin{aligned}\text{add} &= \lambda nmfx.nf(mfx),\\ \text{mult} &= \lambda nmf.n(mf),\\ \text{iszero} &= \lambda nxy.n(\lambda z.y)x,\\ \text{predecesor} &= \lambda n.\lambda f.\lambda x.n(\lambda g.\lambda h.h(gf))(\lambda u.x)(\lambda u.u).\end{aligned}$$

Suppose that we want to define the factorial. We want that:

$$\text{fact}\, n = \text{If}(\text{iszero}\, n)(1)(\text{mult}(n)(\text{fact}(\text{pred}(n))),$$

in orther to do that, look for a fixed point of:

$$\lambda f.\lambda n.\,\text{If}(\text{iszero}\, n)(1)(\text{mult}(n)(f(\text{pred}(n))),$$

and thus fact $= \Theta\lambda f.\lambda n.\,\text{If}(\text{iszero}\, n)(1)(\text{mult}(n)(f(\text{pred}(n)))$. In general:

Definition 3.1.20. Given an stop condition g, an stop value s and a recursive step f, we define the recursive term F that computes (g, s, f) as

$$F = \Theta\lambda f.\,\text{If}(gn)(n)(f(\text{pred}(n))$$

Another implementation of the fixed point operator is *Curry's paradoxical fixed point operator*:

Definition 3.1.21. We define Curry's paradoxical fixed point operator Y as:

$$\text{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)).$$

This operator is used for the Curry's paradox [**sep-curry-paradox**].

3.2 Typed λ-Calculus

Typed λ-calculus is a refinement of untyped λ-calculus, on which the concept of typing is introduced. We are going to present three aproximation to simply typing: *minimal, basic and extended simply typed λ-calculus*.

3.2.1 Definition

We synthesize the definitions of typing presented in [23] or [33]. We structure this definition in three steps: types, terms and equations.

Definition 3.2.1. The types of basic simply typed λ-calculus are built via the BNF:

$$A, B ::= \iota \mid A \to B \mid A \times B \mid 1.$$

where ι denote a basic type.

Remark 3.2.1. We usually refer to basic simply typed λ-calculus as simply typed λ-calculus.

As it happens in sets, where we have that we can consider the set of functions between two set, here we can consider the type of function between two types. This types are called *function types*.

Definition 3.2.2. The *raw terms* of basic simply typed λ-calculus are built via the BNF:

$$A, B ::= x \mid AB \mid \lambda x^t.A \mid \langle A, B\rangle \mid \pi_1 A \mid \pi_2 A \mid * \,.$$

where x denote any variables, and t denote any type.

Definition 3.2.3. We say that a term is closed if it has no free variables.

Remark 3.2.2. We avoid the meticulous redefinition of free and bounded variable and use the one that naturally translates from untyped lambda calculus.

The main difference with untyped calculus, is that every bound variable has a type. That is, for the expression $(\lambda x^t.M)N$ to be coherent we require N to be of type t. This led us to require some sort of condition to N for being of type t.

Nonetheless raw terms have some other meaningless terms, such as projections of a non-pair $\pi_1(\lambda x^t.x)$. We solve all this problems at once with the *Typing rules*. This rules will restricts the semantics of terms. We start by defining *typing context*.

Definition 3.2.4. We state by $M : t$ that a term M is of type t. A typing context is a set of assumption $\Gamma = (x_1 : A_1)$, on which we assume each variable x_i to be of type A_i.

Remark 3.2.3. From this definition, we can state the typing rules, from which we will be able to provide typing. We use the notation $\Gamma \vdash M : A$ meaning that the typing context Γ derives in term M being of type A.

Definition 3.2.5 (Typing Rules)**.** We define the following typing rules, for each typing context Γ.

- Every variable is of the type marked by the context, namely:

$$(var) \qquad \frac{}{\Gamma \vdash x_i : A_i}, \qquad i = 1, .., n.$$

  In addition, $*$ is of type 1.

$$(*) \qquad \frac{}{\Gamma \vdash * : 1}.$$

- From a term of type $A \to B$ we can derive a term of type B, from a term of type A:

$$(app) \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}.$$

  Conversely, we can deduce th

$$(abs) \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A.m : A \to B}.$$

- The projections takes types pairing to each typing:

$$(\pi_i) \qquad \frac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash \pi_i M : A_i}, \qquad i = 1, 2,$$

and conversely:

$$(pair) \qquad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B}.$$

Remark 3.2.4. Not every term can be typed, namely the two previous examples of bad-behaviour: $\pi_1(\lambda x^t.x)$ and $(\lambda x^t.M)N$ where N is not of type t. Thus, in simply typed λ-calculus, we only work with typed terms.

Definition 3.2.6. The terms of *basic simply typed λ-calculus* is a raw typed λ-term that, together with a typing context of every variable, is subject to be typed.

Remark 3.2.5. Typing rules are often called *term-forming operation*. As the only terms that we consider are those suitable to be typed, we can see these typing rules as rules to construct terms inductively.

By assigning type we mean assigning types to the free and bound variables. Further on, when talking about terms we will refer to simply type λ-terms. On these terms, we can define substitution just like we defined it on untyped λ-calculus.

3.2.2 Reductions in simply typed λ-calculus

To talk Church-Rosser, we have to talk reduction. Reductions are basically the same but for the additional structure added that we need to respect:

Definition 3.2.7 (Section 2.5 [33])**.** We define the *single-step β-reduction* as the smallest relationship $\rightarrow_\beta$ such that:

(β): $(\lambda x^t.M)N \rightarrow_\beta M[N/x]$.

$(\beta_{\times,i})$: $\pi_i\langle M_1, M_2\rangle \rightarrow_\beta M_i$.

(cong_1): If $M \rightarrow_\beta M'$ then $MN \rightarrow_\beta M'N$.

(cong_2): If $N \rightarrow_\beta N'$ then $MN \rightarrow_\beta MN'$.

(ζ): If $M \rightarrow_\beta M'$ then $(\lambda x^t.M) \rightarrow_\beta \lambda x^t.M'$.

We define the *multiple-step β-reduction* $\twoheadrightarrow_\beta$ as the reflexive, transitive closure of $\rightarrow_\beta$.

Definition 3.2.8. We define the single-step η-reduction $\rightarrow_\eta$ as the smallest relationship such that:

(η): $(\lambda x^t.Mx) \rightarrow_\eta M$,para todo $x \notin FV(M)$.

(η_1): $\langle \pi_1 M, \pi_2 M\rangle \rightarrow_\eta M$.

$(\eta_\times)$: If $M : 1$ then $M \rightarrow_\eta *$.

(cong_1): If $M \rightarrow_\eta M'$ then $MN \rightarrow_\eta M'N$.

(cong$_2$): If $N \to_\eta N'$ then $MN \to_\eta MN'$.

(ζ): If $M \to_\eta M'$ then $(\lambda x.M) \to_\eta \lambda x.M'$.

Similarly, we define the multiple-step η-reduction $\twoheadrightarrow_\eta$ as the transitive reflexive closure of $\to_\eta$, and the η-equivalence as the symmetric closure of $\twoheadrightarrow_\eta$.

Remark 3.2.6. Note that we basically just add new rules so that β and η reductions pair with the new syntax provided by the product.

We can now talk about about Church-Rosser. Lets check that it does not hold. For example, if $x : A \times 1$ then we can consider $M = \langle \pi_1 x, \pi_2 x \rangle$. Because of the rule η_1, we have that $M \to_\eta \langle \pi_1 x, * \rangle$, but taking $\eta_\times$ into account we can check that $M \to_\eta x$.

Despite this, β-reduction does maintain the Church-Rosser property, so from a computational point of view there is no problem at all.

3.2.3 Minimal and Expanded Typing

Minimal Typing

The already explained λ-calculus can be reduce to a slimmer version. In fact, the only type truly necessary are the function types. We present it succinctly.

Definition 3.2.9. The types of minimal simply typed λ-calculus are built via the BNF:

$$A, B ::= \iota \mid A \to B.$$

where ι denote a basic type.

Definition 3.2.10. The raw term of minimal simply typed λ-calculus are built via the BNF:

$$A, B ::= x \mid AB \mid \lambda x^t.A.$$

where x denote any variables, and t denote any type.

Definition 3.2.11 (Typing Rules)**.** We define the following typing rules, for each typing context Γ.

- Every variable is of the type marked by the context, namely:

$$(var) \qquad \frac{}{\Gamma \vdash x_i : A_i}, \qquad i = 1, .., n.$$

- From a term of type $A \to B$ we can derive a term of type B, from a term of type A:

$$(app) \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}.$$

  Conversely, we can deduce th

$$(abs) \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A.m : A \to B}.$$

Definition 3.2.12. The terms of minimal simply typed lambda calculus are the raw terms that are subject to being typed under a certain Type context.

Remark 3.2.7. We have simply trimmed the fundamental part of the basic definition.

Expanded Typing

We can add to the definition of typing by considering the sum Type. The idea is to have two types A and B and be able to consider a term $\lambda x^{A+B}.x + 1$. That is, we can consider two types to be acceptable by a function.

Definition 3.2.13. The types of expanded simply typed λ-calculus are built via the BNF:

$$A, B ::= \iota | \ A \rightarrow B \mid A \times B \mid A + B \mid 1 \mid 0.$$

where ι denote a basic type.

We proceed to define the raw typed λ-terms.

Definition 3.2.14. The raw terms of expanded simply typed λ-calculus are built via the BNF:

$$\begin{aligned} A, B, C ::= x \mid AB \mid \lambda x^t.A \mid \langle A, B\rangle \mid \pi_1 A \mid \pi_2 A \mid * \\ \mid \text{in}_1\, A \mid \text{in}_2\, A \mid \text{ case } A; x^{t_1}.B \,\text{or}\, x^{t_2}.C \mid \Box^t. \end{aligned}$$

where x denote any variables, and t denote any type.

The term case $A; x^{t_1}.B$ or $x^{t_2}.C$ abstracts the idea of a function on an union term behaves differently in each type that is included. In the typing laws we will see that it is necessary that the type returned by the application of a variable in a function of type union must always converge to the same type (*case* rule).

For this typing, three new rules are included for type derivation, in addition to the previously defined in 3.2.11:

Definition 3.2.15 (Typing Rules)**.** We define the following typing rules, for each typing context Γ.

- Every type is part of a union type including it:

$$(\text{in}_i) \qquad \frac{\Gamma \vdash M : A_i}{\Gamma \vdash \text{in}_i\, M : A_1 + A_2}, \qquad i = 1, 2.$$

  conversely, A term on a union must always have the same type after application.:

$$(\text{case}) \qquad \frac{\Gamma \vdash M : A + B \qquad \Gamma, x : A \vdash N : C \qquad \Gamma, x : B \vdash P : C}{\Gamma \vdash \text{case } A; x^A.M \,\text{or}\, x^B.P : C}.$$

- The $\Box^{\cdot}$ extract a term of any type form the void type.

$$(\Box) \qquad \frac{\Gamma \vdash M : 0}{\Gamma \vdash \Box^A M : A}.$$

We have the same consideration as previously, so that only those being subject to typing will be considered terms. Unification of typing considerations also holds.

Definition 3.2.16. The terms of expanded simply typed lambda calculus are the raw terms that are subject to being typed under a certain Type context.

Pure and impure typed λ-calculus

We have seen that a typing context can vary which terms are subject to be formed and which does not. This consideration will be important later in our discussion, when we consider the category of λ-calculus. In this sense, we can assume that, associated with each typing context, a different lambda computation is generated. This distinction inspires the following definition.

Definition 3.2.17. A basic (resp. minimal, extended) lambda calculus is a triple (Ξ, Λ, Γ) such that:

- Ξ is the collection of types,
- Λ is the collection of terms,
- Γ is a typing context,

where all rules of basic (resp. minimal, extended) lambda calculus are satisfied. When there exists no more types and terms that those derived from the definition, and $\Gamma = \{\}$, it is called a *pure basic (resp. minimal, extended) lambda calculus*.

Remark 3.2.8. Often, in a lambda calculus, is usual to consider terms up to $\alpha\beta\eta$-equivalence.

The way to generate a new lambda calculus, is to create a new type t, (often, based on other types), and to create a (possibly abstract) new term M with typing context $\{M : t\}$.

3.2.4 Unification of typing

We can consider two standards in typing: *Church Style* and *Curry Style*.

- Church style, first shown in [8], is an explicit system, as we do only consider expression with well defined types, in the same way that a function is consider with regard to it domain and codomain. In summary, the typing of a function is the first part of the definition of it, so you can define $f : (0, 1) \to (0, 1)$ and just then define $f(x) = 1/x$. This domain is not it only possible domain, but it what we choose it to be.
- Curry style on the other hand, gain some of the notions of untyped λ-calculus and consider expression as function. To continue the real function analogy, it consider that the function $f(x) = x^2$ just exists, and one can check that it is well defined on $\mathbb{R}$.

More information about the typing style can be found on Chapter 10 and 11 of [17].

Despite this apparent differences, both typing styles can be *unified*. An unifier is a pair of substitution that makes two type styles equal as typed templates. Such an unifier is based onta an algorithm based on type inference. This is instrumental to languages as relevant `python`, that works with implicit typed variables. How this algorithms for unification is shown in great detail in chapter 9 of [33]. Due to this unification, we can consider only explicitely typed terms without any remorse.

3.3 Curry-Howard bijection

3.3.1 Natural deduction

In this section we succinctly introduce the notation of propositional intuitionistic logic, in order to work with them further in the chapter. As we will discuss them in more depth the deduction system in next chapter, we spare our readers of yet another comprehensive introduction of the widely known intuitional propositional logic. Should it be required, more information of propositional logic can be found in, for example, [25] and [37]. Natural deduction originally appears in the works of [14].In this section, we present the more general form.

Thorough this section we replicate notations and process done when defining typed λ-calculus. This is done purposely, as our aim is to proof an equivalence between both system. We start by considering the alphabet consisting a set of countable many variables $x, y, z, ...$ as done previously in lambda-calculus, $\top$ and $\bot$

Definition 3.3.1. The formulas of propositional intuitionistic logic are built via the BNF:

$$A, B ::= x| \ A \to B \mid A \wedge B \mid A \vee B \mid \top \mid \bot.$$

where x denote any variable.

Having the syntax done, is time to provide meaning. We want $\top$ to be the truth value. A formula is true if $\top \to A$[2]. Conversely, a formula is *not-true* or *false* (sometimes denoted as $\neg A$) if $A \to \bot$. In addition, we want all the formulas from which we can derive certain to be true. For this we will define, analogously to the typing-contex from the previous chapter, what a *truth-assumption* is and what the rules of deduction are.

Definition 3.3.2. A truth-assumption is a set of variables $\Gamma = \{x_1, ..., x_n\}$ that we assume to be true. An *judgement* $\Gamma \vdash B$ states that from truth assumption Γ, the formula B can be deduced to be true.

Sometimes we can write a truth assumption Γ, A, that will denote that additionally A is assume to be true. As most times, we will only need a formula A to be true or not without any interest on what exact variable configuration made this possible, so by abuse of notation we can consider $\Gamma = \{x_1, A_1\}$ to have both variables and formulas.

Definition 3.3.3 (Deduction Rules)**.** We define the following deduction rules, for each truth assumption Gamma $\Gamma = \{x_1, ..., x_n\}$.

- Every variable assumed to be true is true:

$$(ax) \qquad \frac{}{\Gamma \vdash x_i}, \qquad \forall i \in 1, .., n.$$

  In addition, $\top$ is always true:

$$(\top) \qquad \frac{}{\Gamma \vdash \top}.$$

- From a true formula $A \to B$ and a true formula A, B can be derived:

$$(\to_1) \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}.$$

[2]check this

Conversely, if assuming A deduces B, then $A \to B$ is true.

$$(\to_2) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash (A \to B)}.$$

- The conjunction being true imply each element to be true:

$$(\wedge_1) \qquad \frac{\Gamma \vdash A_1 \wedge A_2}{\Gamma \vdash A, \qquad \Gamma \vdash B'}$$

and conversely:

$$(\wedge_2) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}.$$

- Every type is part of a union type including it:

$$(\vee_1^i) \qquad \frac{\Gamma_i}{\Gamma \vdash \in_i A_1 + A_2'} \qquad i = 1, 2.$$

conversely, A term on a union must always have the same type after application.:

$$(\vee_2) \qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}.$$

- the *ex falsum quodlibet* (i.e. everything is derivable from falsity) holds:

$$(\bot) \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}.$$

Note that we consider a logic without law of excluded middle. This approach, although somewhat demodé in modern mathematics, was highly popular at the beginning of the twentieth century. It sought to solve the problems of mathematical foundations, in particular the *principle of explosion*.

To work with this logic, we usually work in the usual *truth derivation*. That is, having a formula F, we use the previously introduced rules in other to see whether can they be derivated from the truth assumption. More often than not, we consider the empty assumption, therefore we are working with *tautologies*

Example 3.3.1. We can consider the formula $F = ((x \to y) \wedge (y \to z)) \to ((x \to z)$ and Γ the empty assumption. Then we can deduce:

$$\begin{aligned} \Gamma = \{\} &\vdash ((x \to y) \wedge (y \to z)) \to (x \to z), \\ \Gamma = \{(x \to y) \wedge (y \to z)\} &\vdash (x \to z), \\ \Gamma = \{(x \to y), (y \to z)\} &\vdash (x \to z), \\ \Gamma = \{(y \to z)\} &\vdash (y \to z), \\ \Gamma = \{\} &\vdash \top. \end{aligned}$$

therefore the formula F is true.

As a final note, the definitions of intuitionistic logic and λ-calculus introduced in this work, are made to be matching but are not unique. For example, other source consider that the types of simply typed λ-calculus (instead of the extended typing) and consider only (as is done in [33, Section 6.5]):

$$A, B ::= x|\ A \to B \mid A \wedge B \mid \top.$$

This is called *positive intuitionistic calculus*. In the next chapter we are going to define the concept of *deduction system* and from it, grow into the different approaches to logic.

3.3.2 Bijection

The first approach of λ-calculus to be seen as a deduction system was observed by Curry in 1934[13], early in the development of this area and was completed by Howard in 1980 [18].

Having already asked ourselves when does a term have a type, it is natural to arise the new question: When does a type have a term? This is in fact the fundamental idea of the Curry-Howard isomorphism. For example, considering whether it exists a term for the type$(A \times B) \to B$, is analogue to consider the formula $(A \wedge B) \to B$ to be a tautology. Moreover, the term $\lambda x^{A \times B}.\pi_2 x$ can be seen as a proof of the tautology! we have arrived at the dream of a constructivist mathematician: computational algorithms are demonstrations, and demonstrations are nothing if not algorithms. Lets formalize this intuition.

We can create a pairing between type and formulas by pairing variables with variables and:

Typing name	Types	Formulas
Minimal	Function type $\to$	Implication $\to$
Basic	Type 1 Product type $\times$	$\top$ Conjuction $\wedge$
Expanded	Type 0 Sum type $+$	$\perp$ Disjuction $\vee$

Pairing of formulas and terms.

And we can see that this pairing holds because we can see that Typing rules

- We pair the concept of a formula being true with the concept of a type having a term. We formalize this by pairing truth assumption $\Gamma = \{A\}$ is with the typing context $\Gamma \vdash M : A$.
- We can pair (var) with (ax) and $(*)$ with $(\top)$.
- We can pair (app) with $(\to_1)$ and (abs) with $(\to_2)$.
- We can pair (π_i) with $(\wedge_1)$ and $(\wedge_2)$ with $(pair)$.
- We can pair (in_i) with $(\vee_1^i)$ and $(\vee_2)$ with (case).
- We can pair 0 with $\perp$.

Finally, we pair the terms with the truth derivations. That is, given a term M of type C with associated formula F. By replacing each typing rule needed to deduce

$M : C$ with an associated derivation rule, we can get a truth derivation, and conversely with a truth derivation of $F \rightarrow \top$ from $\Gamma = \{\}$.

From this pairing we can define different ...

3.4 Lambda calculus as a computation system

Around 1930, people start thinking about what mean for a function to be computable. There were three major answer to this question (section 4, [7]):

- In 1930s Alonzo Church introduces the notion of λ-calculus, in which a function is computable if we can write it as a recursive-lambda term onto Church's numerals, that is, if we can deduce the result after a (hopefully finite) deduction steps.
- Later, Alan Turing, who had previously been a doctoral student of Alonzo Church, developed the concept of the Turing machine, in which a function is computable if a tape machine with a limited set of operations is capable of reproducing it.
- Meanwhile, Gödel define the computable function as the minimum set of function such that some properties (such as the existence of a successor function) are includes, and is closed under some operations.

The goal of this three mathematicians was to solve the Entscheidungsproblem[16].

The interest for this type of problems was clear for the mathematical community as the goal was to propose an algorithm that solve every theorem and can be computed. The Church-Turing thesis fromally proved that these three independently generated notions were in fact equivalent[12].

This thesis is also formulated in a philosophical background, to state that every effective computable function system is equivalent to those three. While it can not be formally proven, it has near-universal acceptance to date. Any computing system that can replicate, and thus is equivalent, to either Turing Machines Calculus or λ-Calculus is said to be *Turing Complete*.

Nonetheless, after the formalization of the solution of this problem there were a last problem that was not adverted: the finiteness of time. SAT is the problem that has as input a propositional logic formula and solve whether it is satisfiable. This problem is NP-complete, quite famously the firs of these problems[11]. If we consider the analogous problem for first order algebra, we have an even more complex P-space-complete problem, being almost intractable in worst cases to this date. As a happy consequence for us, much work is left to be done in this area [10].

Chapter 4

Lambeck bijection

In these chapter we raise bridges between Category Theory, λ-calculus, and Propositional logic. The main sources for this chapter were [23], [22] and [33, Chapter 6].

In this chapter we do an important work of unification and organization of the sources used. In general, whenever the connection of category theory and lambda calculus is presented, these previous ones are introduced in their most similar description. We consider that this approach, while requiring less work, loses in knowledge and intuitions.

Avoiding this work means that the bijection does not convey a fundamental idea. A closed Cartesian category naturally extracts the essence of programming. In particular, (following the notation of the definition), it is the adjoint G_3 and the unity of this adjoint that allows us to consider functions and application of functions as elements of the category, thus having a computational system.

In this chapter we will begin by giving the anticipated definition of deductive system, and from this we will be able to give an equational definition of closed Cartesian categories. After this we will define the categories of the various lambda calculi, and finally prove the existence of an adjoint equivalence between the two.

4.1 Deduction systems

Having introduced the notion λ-calculus, category theory and propositional intuitionistic calculus, is time to have a step back, in other to get the full picture.

Definition 4.1.1. A *Deductive system D* is a graph with identity and composition of arrows.

We can reformulated the previous logic with this optic. Prior to that we will make an illustrative example on how is this made. All the properties aforementioned (and paired) show that the requirements of typing/intuitionistic calculus can be expressed in just requirements for the arrows. For example,

Example 4.1.1. 1. Deduction rule $\to_1$: There is an arrow

$$\varepsilon_{A,B} : [B \wedge (B \to A)] \to A.$$

2. From deduction rule $\rightarrow_2$ it can be deduced:

$$\begin{aligned}\Gamma, \{\} &\vdash F = (C \wedge B) \rightarrow A;\\ \Gamma, \{(C \wedge B)\} &\vdash A;\\ \Gamma, \{C, B\} &\vdash A;\\ \Gamma, \{C\} &\vdash B \rightarrow A;\\ \Gamma, \{\} &\vdash F' = C \rightarrow (B \rightarrow A).\end{aligned}$$

That is provided an arrow $h : C \wedge B \rightarrow A$ there is an induced arrow

$$h^* : C \rightarrow [A \rightarrow B].$$

We can check that the second example include an idea, about the relationship between assumption and deduction systems. We can view each assumption of truth as a new deduction system. This also seems logical, since an assumption $\Gamma = \{A_i : i = 1, ..., n\}$ with A_i formulas can be seen as an existence of an arrow $g_i : \top \rightarrow A_i$.

4.1.1 Logical Systems as Deduction systems

In the fashion of previous definitions we can reformulate the logical systems as deduction systems.

Definition 4.1.2. A conjunction calculus is a deductive system D where there is a specified object $\top$ and an operation $\wedge : Ob(D) \times Ob(D) \rightarrow Ob(D)$ such that:

1. For each $A \in Ob(D)$ exists $O_A : A \rightarrow \top$.
2. For each $A_1, A_2 \in Ob(D)$ exists arrows $\pi_i^{A_1,A_2} : A_1 \wedge A_2 \rightarrow A_i$ for $i = 1, 2$.
3. Let $f : C \rightarrow A$ and $g : C \rightarrow B$. Then $\langle f, g\rangle C \rightarrow A \wedge B$.

Note that $\pi_i^{A_1,A_2}$ depend on both A_1, A_2. in the absence of ambiguity we will denote this arrow by π_i.

Definition 4.1.3. A positive intuitionistic calculus is a conjunction calculus D where there is binary an operator $\rightarrow: Ob(D) \times Ob(D) \rightarrow Ob(D)$ such that:

1. For each $A, B \in Ob(D)$, there is an arrow $\varepsilon^{A,B} : [(B \rightarrow A) \wedge B] \rightarrow A$.
2. For each $g : C \wedge B \rightarrow A$ then there is an arrow $g^* : C \rightarrow [B \rightarrow A]$.

Remark 4.1.1. Note that we use [] to avoid ambiguity between $\rightarrow$ as operator or as simply an arrow.

Similarly to π_i, we will usually denote just ε.

Definition 4.1.4. A *intuitionistic propositional calculus* is a positive intuitionistic calculus where there is a specified object $\perp$ and an operation $\vee : Ob(D) \times Ob(D) \rightarrow Ob(D)$ such that:

1. For each $A \in Ob(D)$ there is an arrow $\Box_A :\rightarrow A$.
2. For each $A_1, A_2 \in Ob(D)$ there are arrows $\mathrm{in}_i : A_i \rightarrow A_1 \vee A_2$.
3. For each $A, B, C \in Ob(D)$ there is an arrow

$$h : [(B \rightarrow A) \wedge (C \rightarrow A)] \rightarrow [(C \vee B) \rightarrow A].$$

4.1.2 Categorical systems as Deduction systems

We have already note how a category is a particular case of a deduction system. We can consider how are the logical systems, but we only select those who are also categories.

Remark 4.1.2. Now that we see categories as deduction system, we have a notation problems. We will identify $A \wedge B$ with $A \times B$. While in a deductive system we have define $\langle f_1, f_2 \rangle : C \to A \wedge B$ we have done the same in categories, meaning

$$\langle f_1, f_2 \rangle : C \wedge C \to A \wedge B.$$

This was necessary in deduction system to not put a burden of having a diagonal arrow $C \to C \wedge C$. This make though some extra verbosity in functions like $\langle h\pi_1, \pi_2 \rangle : C \times B \to A \times B$, that would otherwise be the much more legible $\langle h, 1_b \rangle : C \times B \to A \times B$. The consideration with $[f_1, f_2]$ is dual.

Proposition 4.1.1. A Cartesian category $\mathcal{C}$ is a deduction system that is both conjunction system and a category where:

1. $f : A \to T \implies f = O_A$.
2. Given $\langle f_1, f_2 \rangle : C \to A \wedge B$ then $\pi_i \circ \langle f_1, f_2 \rangle = f_i$.
3. Given $h : C \to A \wedge B$ we have that $\langle \pi_1 h, \pi_2 h \rangle = h$.

Proof. Lets begin by proving that a Cartesian category satisfy this conditions. It is clearly a category. Using the specified terminal point $T =$. Considering the product $A \times B = A \wedge B$ for every $A, B \in C$ along with the projections, the others properties are satisfied.

Conversely, considering again $T = \top$ we can check that it is a terminal object. Also, for every $h : C \to A \wedge B$we can see that every h derive the existence of unique f_1, f_2 such that:

$$\begin{tikzcd}
& C \arrow[dl, dashed, bend right, "g"'] \arrow[d, dashed, "f"] \arrow[dr, dashed, bend left, "h"] & \\
A & A \wedge B \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & B
\end{tikzcd}$$

And thus we have finite product. □

Thus, we can start seeing the kind of bijection already shown in the Curry Howard Isomorphism. Hitherto we can adopt the notation $A \times B$ for the conjunction, in the context of Cartesian categories.

Proposition 4.1.2. A closed Cartesian category $\mathcal{C}$ is a deduction system that is a positive intuitionistic calculus that is also a category such that:

1. For all $h : c \times b \to a$, we have that $\varepsilon\langle h^* \pi_1, \pi_2 \rangle = h$.
2. For all $k : c \to [b \to a]$, we have that $(\varepsilon\langle k\pi_1, \pi_2 \rangle)^* = k$.

Proof. Lets begin checking that $\mathcal{C}$ is a positive intuitionistic calculus and holds 1. and 2. As it is a Cartesian category, it is a conjunction calculus.

- We identify $a \to b$ with the exponential object b^a. For every $b \in Ob(C)$, following the notation from definition 2.2.6, we have an adjoint $F_3^b \dashv G_3^b = (F^b, G^b, \varphi^b, \eta^b, \varepsilon^b)$. Then, we define the arrows from definition 4.1.3 as

$$\varepsilon^{a,b} : [(b \to a) \wedge b] \to a = \varepsilon^b(a), \qquad g^* = \varphi(g).$$

- Then, 1. is only consider that, given an $h : F_3^b c = c \times b \to a$, we have that, $\varepsilon^b(a) \circ F_3^b(\varphi(h)) = h$, due to the fact that

$$(\varphi^b)^{-1}(h : d \to Gc) = \varepsilon^b(c) \circ F_3^b(h).$$

- Conversely, if part 1. was the left part of the inverse definition, 2. is the right part.

To prove that any intuitionistic calculus that is also a category is a ccc, we just made the same pairing, $a \times b = a \wedge b$ and $a \to b = b^a$ to define F_b^3 on both objects and arrows, (where $F_b^3(h : a \to a') = \langle h, 1_b \rangle$), define G_3^b on objects, and define $\varepsilon^b(a)$ as $\varepsilon^{a,b}$.

□

These propositions allow us to easily define graph completions to generate a to generate from these CCCs and positive conjunction calculus.

Proposition 4.1.3. A *closed Cartesian category* is a deduction system that is both a intuitionistic propositional calculus and a category, with the additional equations:

1. $f : \bot \to A \implies f = \Box_A$.
2. Given $[f_1, f_2] : A \vee B \to C$ then $[f_1, f_2] \circ \mathrm{in}_i = f_i$.
3. Given $h : A \vee B \to C$ we have that $[h\,\mathrm{in}_1, h\,\mathrm{in}_2] = h$.

Remark 4.1.3. Is usual to denote $A \vee B$ as $A + B$ or $A \sqcup B$, as we are working with the coproduct.

Proof. We have proved that it is already a closed Cartesian category. Extra considerations are dual to does already considered for a Cartesian category. □

4.1.3 Logic in closed Cartesian categories

We want now to introduce an idea of logic implications in the context of a closed Cartesian category C. For that, we can use the naive idea to consider the terminal object $\top \to A$, as a value of truth, and then deduce that any object $a \in C$ such that there exists an arrow $f : \top \to a$, then A is also true. In this idea, the natural progression is to consider a a to be true by desire, so we add a new arrow $x : \top \to a$ to the category, and complete the category so the axiom of metacategories are still satisfied. In this section we are going to formalize this idea.

Definition 4.1.5. Given a (Cartesian, closed Cartesian) category A, we say that an *indeterminate* arrow $x : a \to b; a, b \in A$ is an arrow $x : a \to b$ not necessarily included in $Ar(A)$. Indeterminates are also called *assumptions*.

Remark 4.1.4. Whenever we talk about an indeterminate arrow, we just talk about an arrow with the special quality that it does not have to exist within our predefined set of arrows. The new emphasis is that we do not care what arrow it is exactly, in a similar way as it is usually do with numeric variables.

Definition 4.1.6. [23, Part I, Chapter 5] Given a (Cartesian, closed Cartesian) category A, and a indeterminate $x : a \to b$, we define the *category of A based on the indeterminate $x : a \to b$*, denoted by $A[x]$, as the category that have:

1. As objects the same object as in A.
2. As arrows it has the indeterminate x in addition to $Ar(A)$ and all the needed combination between those two, under the equivalence relationship:
   - Preserve A-equivalence: for all $g : b \to c, f : a \to b, h : a \to c \in A$,
   $$gf = h \in A \qquad \text{implies} \qquad gf \cong h \in A[x].$$
   - Preserve composition: for all $f : a \to b, f' : a \to b, g : b \to c, g' : b \to c \in Ar(A[x])$,
   $$f \cong f', g \cong g' \qquad \text{implies} \qquad fg \cong f'g'.$$
   - Preserve identity: for all $f : a \to b \in A[x]$ we have $f1_a \cong f \cong 1_b f$.
   - Associative: for all $f : a \to b, g : b \to c, h : c \to d$ we have $(fg)h \cong f(gh)$.

Thus arrows are equivalence classes. Arrows in $A[x]$ are called *polynomials* in x.

Remark 4.1.5. We define the equivalence on arrows $\cong$ so that the inclusion $A \hookrightarrow A[x]$ is a functor.

Definition 4.1.7. Let A be a (Cartesian, closed Cartesian) category and $X = \{x_1 : a_1 \to b_1, ...x_n : a_n \to b_n\}$ be a set of assumptions. We define the category $A[X]$ as $A[x_1][x_2]..[x_n]$.

Remark 4.1.6. Note that for every permutation $\sigma : \{1, ..., n\} \to \{1, .., n\}$, we have that $A[x_1]...[x_n] \cong A[x_{\sigma(1)}]...[x_{\sigma(n)}]$.

Proposition 4.1.4. Let C, C' be (Cartesian, closed Cartesian) categories, F be a (Cartesian, closed Cartesian) Functor, and $X = x_1, ..., x_n, X' = x'_1, ..., x'_n$ be sets of assumptions for C and C' respectively. Then there is an unique (Cartesian, closed Cartesian) Functor $F_{x_1,...,x_n}$ such that $F(x_i) = x'_i$ the following diagram commutes.

$$\begin{tikzcd}
C[X] \arrow[r, dashed, "F_{X \to X'}"] & C'[X] \\
C \arrow[u, hook] \arrow[r, "F"] & C' \arrow[u, hook]
\end{tikzcd}$$

Proof. Immediate as $C[x_1, ..., x_n]$ is the expansion of C and $x_1, ..., x_n$. □

4.2 Categorical understanding of simply typed λ-calculus

4.2.1 Natural Numbers

We begin this section by motivating the introduction of natural numbers. The importance of these is that we want a computation system. These systems, in a modern practical case, are actually intended to compute a function from the natural numbers to the natural numbers (either on a tape in a Turing machine, a recursion as in Gödel's computational system, or an abstract formalization in λ-calculus).

We have already seen how natural numbers can be done in untyped calculus. However in this section we will approach the problem, formalizing the notion for typed calculus and introducing the concept of natural numbers in categories.

Natural Numbers in Categories

Definition 4.2.1. Given a closed Cartesian category C, $(N, S : N \to N) \in Ob(C) \times Ar(C)$ is said to be a *natural number* when, for every $f : A \to A \in C$, there is an unique h such that

$$\begin{tikzcd} 1 \arrow[r, "0"] \arrow[dr, "a"'] & N \arrow[r, "S"] \arrow[d, dashed, "h"] & N \arrow[d, dashed, "h"] \\ & A \arrow[r, "f"] & A \end{tikzcd}$$

When h exists but is not necessarily unique (N, S) is said to be a *weak natural number*.

We can consider natural numbers as an initial object. Having a closed Cartesian category C, we can consider it as a marked category C_*, such as in Set_*, with the specified object being $\top = 1$. We can also consider the category J provided by the diagram:

$$1 \xrightarrow{0} N \xrightarrow{S} N$$

with the specified point being 1. Then considering C_*^J we can see that every arrow is such that:

$$\begin{tikzcd} 1 \arrow[r, "0"] \arrow[d, "1_1"] & N \arrow[r, "S"] \arrow[d, "h"] & N \arrow[d, "g"] \\ 1 \arrow[r, "a"] & A \arrow[r, "f"] & A \end{tikzcd}$$

and is denoted as $(1, h, g)$. Considering the category $(C_*^J)'$ with objects as in C_*^J and arrows $\{(1, h, g) \in C_*^J : h = g\}$ we have that a natural number for C is an initial object in $(C_*^J)'$. This observation allow us for a simple proposition as a consequence of proposition 1.2.2:

Proposition 4.2.1. Any two natural number objects are isomorphic.

Every time natural are talked, induction have to appear in some part. The intuition of the categorical concept is that we have a object N that is the set of every natural, and a successor arrow S.

Definition 4.2.2 (Category of closed Cartesian categories)**.** We can define the *category of closed Cartesian categories with natural numbers $CCart_N$* that has as objects all small ccc, and as objects all functors that preserve the structure of closed Cartesian categories and added natural numbers structure. Maybe add the proposition and corollary 9.1 and 9.2 of Lambeck.

Natural Numbers in Typed λ-calculus

As a consequence of the Curry-Howard isomorphism we have the identification between logic and lambda-calculus. Afterwards, as we start consider different deduction system (conjunction calculus, positive, intuitionistic) in logic, we can consider different typed λ-calculus(basic, standard, expanded), each one paired with a deduction system. Finally, we restricted these deduction systems to the world of categories these gave us also another set of paired categories (Cartesian, closed Cartesian, closed bi Cartesian).

Having laid the intuition of what is a natural number in category theory we can proceed to explain what a natural number is in Typed λ-calculus. Relating the notions of typed λ-calculus shown in [23, Section 10], [33, Section 6] and [7, Section 10], we can see that it is opted by Lambek to include to simply include some structure over the basic type ι to provide with a natural number type:

Remark 4.2.1. Remember that ι was a basic type with no added structure.

Definition 4.2.3 (Lambek natural numbers)**.** *Natural number in simply typed lambda-calculus* is a simply typed lambda calculus with the added structure for raw typed λ-terms:

$$A, B, C ::= ... \mid \mathfrak{o} \mid S(A) \mid I_t(A, B, C);$$

for every type T, and the related typing rules:

- If N is of type ι so is $S(N)$:

$$(S) \qquad \frac{\Gamma \vdash N : \iota}{\Gamma \vdash S(N) : \iota}.$$

  In addition, $\mathfrak{o}$ is of type ι:

$$(\mathfrak{o}) \qquad \frac{}{\Gamma \vdash \mathfrak{o} : \iota}.$$

- If $M : A$, $H : A \to A$ and $N \in \iota$ then we can apply H to M, N times and get a result in A:

$$(I) \qquad \frac{\Gamma \vdash N : \iota, \qquad \Gamma \vdash M : A, \qquad \Gamma \vdash H : A \to A}{\Gamma \vdash I_A(M, H, N) : \iota}.$$

Remark 4.2.2. Natural numbers can be define over any of the different typed λ-calculus. Thus we made a definition by addition of structure.

This option has some positive notes:

1. We have, as in categories, an object N that is a natural number.
2. It encapsulate in formalism the properties of Church numerals in untyped λ-calculus.

Nonetheless we have already carried out a construction over natural numbers in untyped λ-calculus. That construction is replicable in the context of typed λ-calculus.

Definition 4.2.4 (Church numerals in typed lamdba-calculi)**.** Let We define the nth Church numeral in typed λ-calculi as, for a type t:

$$\overline{n}^t = \lambda f^{t \to t}.\lambda x^t.f^n(x).$$

Remark 4.2.3. $\overline{n}^t$ is of type $(t \to t) \to (t \to t)$.

As a consequence of this proposition, we can consider a type N^t such that $\overline{n}^t : N^t$ for every $n \in \mathbb{N}$ and every type t. Nonetheless:

1. To have a similar behaviour of multiple application of a function, the natural numbers are dependent on the type. So to speak, there is a natural number family for each type t.
2. There is no type exclusively for natural numbers, as not necessarily every element of $(t \to t) \to (t \to t)$ is a natural number.

3. On the plus side, we have the tools to make both a term S and a term I_t in the same manner as it was done in untyped λ-calculus.

From this point on, we will use the Lambek description of natural numbers, letting Church numerals as a construction for untyped λ-calculus.

Lastly, we can consider the effect of Lambeck description in β-reduction.

4.2.2 Category of typed λ-calculus

The objective of this section is to finish forming an idea that has been present for some time in this work: the existence of several λ-calculations.

In chapter 3, we have described how there exists different λ-calculus, the same way that there exists multiple groups, as either lambda calculus and a group is only a structure,. We may continue by defining the category of λ-calculus. For that we first define the concept of morphism as a structure preserving mapping.

Definition 4.2.5 (Morphisms of lambda-calculus)**.** Let Λ, Λ' be minimal (resp. basic, expanded) λ-calculus (resp. with natural numbers). A morphism $F : \Lambda \to \Lambda'$ is a function that

1. Maps types in Λ to types in Λ', Terms in Λ to terms in Λ' and if $a \in \Lambda$ is a closed term, so is $F(a)$, and sends variables to variables. For example:

$$F(1) = 1, \qquad F(*) = * \qquad or \qquad F(A + B) = F(A) + F(B).$$

2. Preserves the structure of minimal (resp. basic, expanded) λ-calculus (resp. with natural numbers) up to α-equivalence, β-equivalence and η-equivalence.

Morphisms of λ-calculus will be also called λ-morphism for short, in cases of ambiguity.

Definition 4.2.6 (Category of lambda-calculus)**.** We can define the category of minimal (resp. basic, expanded) typed λ-calculus as the category that:

- Have as objects the different minimal (resp. basic, expanded) typed λ-calculus.
- Have as arrow the λ-calculus morphism.

Similarly, we can define the category of minimal (resp. basic, expanded) with natural numbers as in remark 4.2.2, adding that morphism should also preserve natural numbers.

4.2.3 The internal language of a Closed Cartesian Category

Up to this moment, we have related that a closed Cartesian category is a positive intuitionistic calculus, that is equivalent to typed λ-calculus, with some extra structure.

Now we have to consider the original ideas that are consider to be considered the precursor of categorical thinking: the poincare functor.l This functor is, in short, sees how to consider a group within a topology, and see that we can infer group morphism from topologies homeomorphism. This is the original idea behind the functors, and we will now provide an example that is similar in flavor to the poincare functor: we will generate a *lambda*-calculus from a CCC.

Definition 4.2.7. Let C be a closed Cartesian category. Then its *internal language* $\mathcal{L}(C)$ is the λ-calculus that:

- Has as types the objects of C, with function types generated by exponential types, conjunction types by product types, and the 1 type by the terminal object $\top \in Ob(C)$.
- In $\mathcal{L}(A)$ a variable x^{t_i} of type t is just an indeterminate $x : 1 \to t_i$, and terms of type A are just a polynomial $\varphi(x_1^{t_1}, ..., x_n^{t_n}) : 1 \to A$ over indeterminate $x_i : 1 \to t_i$, for i in $1, ..., n$.

Remark 4.2.4. In any simply typed λ-calculus, we consider raw terms and then we consider only those that are subject to be typable. This, was later understood as any term being a proof of veracity of a type, via the Curry-Howard isomorphism. So, to maintain this idea of the arrows of the language being proof of veracity, we see them as arrows $f : \top \to A$, that is, as deduction from truth.

Now, we have to define how the functor behave over morphism.

Definition 4.2.8. Let $F : C \to C' \in CCart_N$. We define $\mathcal{L}(F)$ as the λ-morphism such that:

- $\mathcal{L}(F)(a) = F(a)$ for every term a, that is, for every $a \in Ob(C)$.
- $\mathcal{L}(x^a) = x^{F(a)}$ for every variables $x^a \in \mathcal{L}(C)$, that is, for every assumption $x : 1 \to A$ in C.
- Let $X = \{x_1^{t_1}, ..., x_n^{t_n}\}$ be a set of indeterminates. Then for ever polynomial φ over X we have that

$$\mathcal{L}(F)(\varphi(X) : 1 \to A) = F_{X \to \mathcal{L}(X)}(\varphi(X) : 1 \to A)$$

  where $F_{X \to \mathcal{L}(X)}$ is defined as in proposition 4.1.4.

Proposition 4.2.2. If C has weak natural numbers, $\mathcal{L}(C)$ has natural numbers.

Proof. □

Proposition 4.2.3. If $F : C \to C \in CCart_N$, then $\mathcal{L}(F)$ maintain natural numbers.

4.2.4 Cartesian Closed Category generated by a typed *lambda*-calculus

Given a typed λ-calculus, we are going to construct a ccc conversely to the process done previously.

Definition 4.2.9. Given a typed lambda calculus L we can construct a *generated closed Cartesian category*, denoted as $\mathcal{C}(L)$ as follow:

- As object it has the types in L.
- As arrows $f : a \to b$ it has the lambdas $\lambda x^a.b(x)$ where $b(x)$ is a term that may have x^a as a free variable.
- Arrows are considered as equivalence classes up to $\alpha\beta\eta$-equivalence.

The following proposition is discussed in plenty of sources. We choose adapt the approximation done by [31] to dependent typing, instead of following the main source [22] as we consider that it has a more natural approach.

Proposition 4.2.4. $\mathcal{C}(L)$ is a Cartesian closed category.

Proof. We start checking the category axioms:

- Identity is defined by $\lambda x^a.x$.
- Composition of $\lambda x^a.f(x) : a \to b$ and $\lambda x^b.g(x^b) : b \to c$ then the composition is defined by $\lambda x^a.g(f(x)) : a \to c$.
- Associativity follows from β-equivalence.

To provide the terminal object:

- The type 1 is the terminal object. By η_x the only term of type 1 is $*$ up to equivalence. Therefore, any arrow $t : a \to 1$ must be equivalent to $\lambda x^a.*$.

To provide the product type for two type a, b.

- The type product is provided by $a \times b$.
- The projections are provided by $\beta_{x,i}$ with the projections π_1, π_2.
- Given three types a, b, c and lambdas $\lambda x^a.f(x) : a \to b$, $\lambda x^a.g(x) : a\ toc$.
- Properties 2. and 3. in proposition 4.1.1 comes directly from η_x and $\beta_{x,i}$ equivalence rules.

To provide the exponential type (and thus have a conjunction calculus):

- The exponential type a^b is $b \to a$.
- Given $h = \lambda x^{a \times b}.f(x) : a \times b \to c$ we define $h^* = \lambda x^a.\lambda y^b.f(\langle x, y\rangle)$.
- There is an arrow $\varepsilon^{a,b} = \lambda y^{a \to b \times a}.(\pi_1(y))(\pi_2(y))$.
- TODO:Check this. Properties 1. and 2. in proposition 4.1.2 comes from β and ζ equivalence rules.

$\square$

Proposition 4.2.5. If L has a natural numbers structure, then $\mathcal{C}(L)$ has a natural number structure.

Now we repeat the process as with $\mathcal{L}$ and define a functor $\mathcal{C} : \lambda\text{-Calc} \to CCart$.

Proposition 4.2.6. Let $L \in Ob(\lambda\text{-Calc})$, $\mathcal{C}(L)$ be the generated closed Cartesian category of L and let $F : L \to L' \in \lambda\text{-Calc}$. We define the functor $\mathcal{C}(F)$:

- $\mathcal{C}(F)(a) = F(a)$ for every $a \in Ob(C)$.
- $\mathcal{C}(F)(\lambda x^a.b(x) : a \to a') = \lambda x^{F(a)}.F(b(x))$.

Proof. $\square$

4.2.5 Equivalence

After defining in the two previous sections we define two functors/ the language of a closed Cartesian category as a functor $\mathcal{L}$ and the closed Cartesian category of a lambda calculus $\mathcal{C}$. We now will proof that these two functors makes an adjoint equivalence.

Chapter 5

Locally Cartesian closed categories

In this chapter we introduce the notions of [31]. In this we use hyperdoctrines as a tool to better understand closed Cartesian categories and relate them with a new typing theory: Martin-Lof type theory.

5.1 Martin-Löf type theory

Appendix A

Haskell

A.1 BNF Notation in Haskell

Acknowledgements

Bibliography

[1] Rod Adams. *An Early History of Recursive Functions and Computability: From Gödel to Turing*. Docent Press, 2011.

[2] Jesse Alama and Johannes Korbmacher. "The Lambda Calculus". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Summer 2021. Metaphysics Research Lab, Stanford University, 2021.

[3] Nlab Authors. *Stone-Cech compactification,* [Online; accessed 30-June-2021]. 2012. URL: http://ncatlab.org/nlab/revision/Stone-%C4%8Cech%20compactification/8.

[4] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1999.

[5] Stefan Banach. "Théorie des opérations linéaires". In: (1932).

[6] Ronald Brown. "Groupoids and van Kampen's theorem". In: *Proceedings of the London Mathematical Society* 3.3 (1967), pp. 385–401.

[7] Felice Cardone and J Roger Hindley. "History of lambda-calculus and combinatory logic". In: *Handbook of the History of Logic* 5 (2006), pp. 723–817.

[8] Alonzo Church. "A formulation of the simple theory of types". In: *The journal of symbolic logic* 5.2 (1940), pp. 56–68.

[9] Alonzo Church. "A set of postulates for the foundation of logic". In: *Annals of mathematics* (1932), pp. 346–366.

[10] Stephen Cook. "The P versus NP problem". In: *The millennium prize problems* (2006), pp. 87–104.

[11] Stephen A Cook. "The complexity of theorem-proving procedures". In: (1971), pp. 151–158.

[12] B Jack Copeland. "The church-turing thesis". In: (1997).

[13] Haskell B Curry. "Functionality in combinatory logic". In: *Proceedings of the National Academy of Sciences of the United States of America* 20.11 (1934), p. 584.

[14] Gerhard Gentzen. "Untersuchungen über das logische Schließen. I". In: *Mathematische zeitschrift* 39.1 (1935), pp. 176–210.

[15] HaskellWiki. *Hask — HaskellWiki,* [Online; accessed 9-June-2021]. 2012. URL: https://wiki.haskell.org/index.php?title=Hask&oldid=52908.

[16] David Hilbert and Wilhelm Ackermann. *Principles of mathematical logic*. Vol. 69. American Mathematical Soc., 1999.

[17] J Roger Hindley and Jonathan P Seldin. *Lambda-calculus and Combinators, an Introduction*. Vol. 13. Cambridge University Press Cambridge, 2008.

[18] William A Howard. "The formulae-as-types notion of construction". In: *To HB Curry: essays on combinatory logic, lambda calculus and formalism* 44 (1980), pp. 479–490.

[19] Daniel M Kan. “Adjoint functors”. In: *Transactions of the American Mathematical Society* 87.2 (1958), pp. 294–329.

[20] Donald E Knuth. “Backus normal form vs. backus naur form”. In: *Communications of the ACM* 7.12 (1964), pp. 735–736.

[21] Kenneth Kunen. *Set theory an introduction to independence proofs*. Elsevier, 2014.

[22] Joachim Lambek. “Cartesian closed categories and typed λ-calculi”. In: *LITP Spring School on Theoretical Computer Science*. Springer. 1985, pp. 136–175.

[23] Joachim Lambek and Philip J Scott. *Introduction to higher-order categorical logic*. Vol. 7. Cambridge University Press, 1988.

[24] Saunders Mac Lane. *Categories for the working mathematician*. Vol. 5. Springer Science & Business Media, 2013.

[25] Victor W Marek. *Introduction to mathematics of satisfiability*. CRC Press, 2009.

[26] Bartosz Milewski. *Category theory for programmers*. Blurb, 2018.

[27] Barry Mitchell. *Theory of categories*. Academic Press, 1965.

[28] James R Munkres. *Topology. Featured Titles for Topology Series*. 2000.

[29] Emily Riehl. *Category theory in context*. Courier Dover Publications, 2017.

[30] Joseph J Rotman. *An introduction to homological algebra*. Springer Science & Business Media, 2008.

[31] Robert AG Seely. “Locally cartesian closed categories and type theory”. In: *Mathematical proceedings of the Cambridge philosophical society*. Vol. 95. 1. Cambridge University Press. 1984, pp. 33–48.

[32] Herbert Seifert. *Konstruktion dreidimensionaler geschlossener Räume*. Hirzel, 1931.

[33] Peter Selinger. “Lecture notes on the lambda calculus”. In: *arXiv preprint arXiv:0804.3434* (2008).

[34] Sigur. *Using Tikz-cd to Illustrate Composition of Natural Transformations*. Computer Science Stack Overflow. URL: https://tex.stackexchange.com/questions/505322/using-tikz-cd-to-path-to-illustrate-composition-of-natural-transformations%20(version:%202021-02-05).

[35] Alan Mathison Turing. “On computable numbers, with an application to the Entscheidungsproblem. A correction”. In: *Proceedings of the London Mathematical Society* 2.1 (1938), pp. 544–546.

[36] Egbert R Van Kampen. “On the connection between the fundamental groups of some related spaces”. In: *American Journal of Mathematics* 55.1 (1933), pp. 261–267.

[37] Philip Wadler. “Propositions as types”. In: *Communications of the ACM* 58.12 (2015), pp. 75–84.

[38] Kinoshita Yoshiki. “Prof. Nobuo Yoneda passed away”. In: (1996). URL: https://www.mta.ca/~cat-dist/catlist/1999/yoneda.